MASTER EN SCIENCES INFORMATIQUES

E.A.O. projet d'aide à la factorisation de polynômes

Lambert, Marc

Award date:
1985

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix
(Namur)

Institut d'Informatique

E.A.O.
PROJET D'AIDE A LA FACTORISATION DE POLYNOMES

Mémoire présenté par
Marc LAMBERT
en vue de l'obtention du titre
de licencié et maître en informatique.
Promoteur Monsieur Claude CHERTON.

Année académique 1984-1985.

REMERCIEMENTS

Avant tout, je tiens a remercier Monsieur Cherton pour sa disponibilité et pour son enseignement.
Je remercie Monsieur Collinet pour la documentation précise et complète qu'il a bien voulu écrire.
Merci à mon épouse pour sa patience, ses encouragements et pour l'édition des annexes.

TABLE DES MATIERES

Chapitre I INTRODUCTION

Chapitre II

ANALYSE DE LA FACTORISATION

Chapitre III

CONSTRUCTION D'UN ALGORITHME DE FACTORISATION

Chapitre IV

CONSTRUCTION DES STRUCTURES DE DONNEES

Chapitre V

IMPLEMENTATION DE LA MISE EN EVIDENCE

CONCLUSION

ANNEXE

1. Annexe1: Définition et exercices typiques
2. Annexe2: Scénario
3. Annexe3: Erreurs habituelles
4. Annexe4: Spécification et code de la mise en évidence
5. Annexe5: Manuel utilisateur

BIBLIOGRAPHIE

"Ils croient atteindre au scientifique parce qu'ils quantifient. Ce qu'ils inventent par le modèle devient réalité, le monde réel, la vérité et ils ne voient pas la véritable 'perversion' dont ils sont l'objet". Eric Deleersnyder.

Chapitre I

INTRODUCTION

1. Choix du sujet
2. Besoins de l'enseignant
3. Les grandes fonctions du didacticiel
4. Précisons le sujet
    - 4.1. Faire de l'enseignement
    - 4.2. La boîte à outils
    - 4.3. Faire de l'informatique
5. E.A.O.
    - 5.1. Types d'E.A.O.
    - 5.2. Comment concevoir notre didacticiel
6. Notre démarche

Ce mémoire contient la première étape de la réalisation d'un projet en E.A.O. visant à apprendre la factorisation de polynômes.

Ce premier chapitre précise nos objectifs informatiques et pédagogiques. Il dégage la démarche que nous avons suivie.

1.LE CHOIX DU SUJET

A l'époque du choix de mon mémoire, je me destinais à l'enseignement. Je voulais enseigner les mathématiques et l'informatique. Dès lors, quoi de plus naturel que d'utiliser l'informatique pour assister l'enseignement de matières étudiées pendant six ans. Lors de la première réunion avec Monsieur Cherton, j'ai découvert en l'E.A.O. un terrain où convergeaient les mathématiques, l'enseignement et l'informatique.

2.BESOINS DE L'ENSEIGNANT

Ce mémoire est le fruit d'une collaboration active avec Monsieur Collinet, professeur de mathématique. Celui-ci nous a exposé ses besoins, à savoir, disposer d'un programme sur micro-ordinateur permettant d'apprendre (ou réapprendre) la factorisation systématique de polynômes à des étudiants de dernière et avant dernière année d'humanités. Ceci pour éliminer tout problème dans d'autres applications plus générales qui font appel à la recherche de racines de polynômes.

3.LES GRANDES FONCTIONS DU DIDACTICIEL

D'une première analyse des besoins et du scénario proposé par monsieur Collinet, nous avons retiré les grandes fonctions à réaliser. Le produit final devra

1.proposer des exercices et des exemples de niveaux de difficulté différents;

2.détecter les erreurs, les diagnostiquer et susciter une réflexion sur l'erreur commise par l'élève;

3.proposer une ou plusieurs alternatives à la solution donnée par l'élève;

4.factoriser purement et simplement en indiquant la démarche suivie;

5.évaluer la démarche de l'élève dans sa résolution d'un exercice;

6.proposer un interface agréable:
- écran clair
- animation;

7.orienter un élève en difficulté;

8.assurer un suivi de l'élève.

4.PRECISONS LE SUJET

4.1.Faire de l'enseignement

Le didacticiel devra offrir un dialogue proche de celui qui existe entre un professeur et un étudiant. Il devra orienter, expliquer une démarche et non indiquer directement une solution. Il devra aider à découvrir, expliquer les erreurs et non déclarer sèchement leur existence.

4.2.La boîte à outils

Ce mémoire s'insère dans un projet de plus grande envergure. Les didacticiels actuellement sur le marché ne rencontrent pas les besoins spécifiques des enseignants. Les enseignants informaticiens, pour leur part, ne voient pas leur intérêt à consacrer 100, 150 heures (pour un cours programmé) de mise au point d'un didacticiel qui les aideraient à donner 1 heure de cours. Certains n'en ont d'ailleurs pas les capacités. D'autre part, les enseignants utilisent souvent plusieurs manuels de référence de base desquels ils tirent leur enseignement.

Dans cet ordre d'idée , il est intéressant de leur proposer non pas un didacticiel rigide et coûteux en heures de préparation mais une boîte à outils adaptable à leurs besoins pédagogiques. Nous nous proposons donc, de construire des modules de base dont l'assemblage est laissé à l'entière liberté de l'utilisateur enseignant. Celui-ci devra connaître un minimum d'informatique pour construire son cours avec les outils que nous lui proposerons.

De plus, il semble nécessaire de fournir, avec la boîte à outils, un ou plusieurs didacticiels "tout faits" pour initier le futur utilisateur à l'assemblage des modules de base.

4.3.Faire de l'informatique

Nous n'avons pas la prétention de réaliser en un an l'entièreté de la boîte à outils "d'aides à la factorisation". Deux ou trois mémoires seront nécessaires pour arriver à une boîte à outils opérationnelle. Nous avons, dès le début, décidé de nous attaquer d'abord à la réalisation d'un sous-système, celui de factorisation pure et simple de polynômes.

La réalisation d'un sous-système et ce, dans une démarche générale de conception constitue l'objectif informatique de ce mémoire. Nous développons cet objectif dans les deux paragraphes suivants.

5.L'E.A.O.

Digit, bureautique, robotique, télèmatique, tèlèinformatique, autant de mots qui font peur. Il est important de démystifier l'informatique. Commençons donc à l'école, en utilisant l'ordinateur comme outil de la même manière que livres, tableaux et autres matériels didactiques mais aussi en essayant d'aller plus loin, au maximum de ses possibilités. Apprenons aux étudiants, aux enseignants de tout âge, comment utiliser l'ordinateur, comment il fonctionne. Il faut qu'ils domptent cette machine nouvelle, qu'ils en connaissent les possibilités et les limites, qu'ils connaissent les modes de raisonnement de l'informatique.

Démystifions l'informatique aux yeux des enseignants. De ma propre expérience, il y a baucoup d'enseignants avec qui il faut éviter de parler informatique notamment en primaire. Ceux-ci redoutent le nouveau "monstre", doutent de ses possibilités pédagogiques et appréhendent son introduction dans les écoles. Beaucoup croient encore que l'informatique à l'école signifie la disparition du maître. Donc, enseignons l'informatique aux enseignants et aux enseignés; assistons l'enseignant par l'ordinateur.

5.1.Types d'E.A.O.

Nous entendons par E.A.O. toute utlisation de l'ordinateur comme outil pédagogique.

On peut classer les différents types d'E.A.O. suivant l'utilisation qu'on fait de l'ordinateur. La liste qui suit n'est pas exhaustive. Nous voulons simplement citer quelques utilisations typiques.

D'abord, il y les didacticiels qui proposent des exercices, les corrigent et évaluent les réponses de l'élèves. Il y a les didacticiels d'enseignement programmé qui contiennent de la matière à assimiler par l'élève, un contrôle de la compréhension par questions-réponses et une succession de modules suivant le niveau des réponses de l'élève. Il y a les "bibliothèques", didacticiels de documentation, d'illustration. Il y a les simulateurs par exemple, les didacticiels qui simulent des phénomènes physiques ou chimiques. Remarquons que les quatre types cités peuvent se mélanger dans la même application.

5.2.Comment concevoir notre didacticiel

Nous pouvions nous contenter de mémoriser un certain nombre d'exercices ainsi que leurs démarches de solution; ensuite, puiser dans cette réserve à la demande. Nous pouvions utiliser un langage d'auteur. Celui-ci est basé sur des concepts de mise en pages, d'affichage à l'écran, de tests. Tout est explicitement prévu dans le didacticiel: il contient, par exemple, une série d'exercices avec leurs solutions et leurs listes d'erreurs; chaque nouvel exercice s'accompagne d'une même démarche d'insertion.

Mais, notre objectif est de concevoir un programme "intelligent" qui, par exemple génère lui-même les exercices. Ce programme devra assurer un dialogue entre la machine et l'élève. Pour assurer ce dialogue relatif à la démarche de factorisation, nous devons créer un programme qui reflète la démarche désirée chez l'élève. Notre programme simulera la tâche intelligente de factorisation. Macroscopiquement, le programme utilisera les mêmes mécanismes que l'être humain.

6. NOTRE DEMARCHE

Pour maîtriser le dialogue élève-machine, nous devons connaître les mécanismes de factorisation, maîtriser les méthodes de factorisation.

Le chapitre II contient notre analyse de la factorisation à partir de ce que nous connaissions déjà et telle que nous l'a apprise monsieur Collinet par les documents qu'il nous a remis (v. Annexe 1: définition et exercices typiques). Ce chapitre II contient aussi les conclusions de notre analyse: les catégories de méthodes, les définitions des expressions manipulées, la stratégie de factorisation.

L'objectif suivant que nous nous sommes fixé est de représenter les expressions sur lesquelles on applique les méthodes. Il faut construire des structures de données dans un souci constant de performance. Les structures de données doivent être telles que la rapidité d'action des primitives utilisées permettent un temps de réponse de factorisation raisonnable; telles que la place mémoire occupée soit raisonnable et possible sur Apple II; telles que les algorithmes de factorisation soient relativement simples.

Nous avons alors utilisé la démarche de construction suivante:

1. d'abord, construire un algorithme de factorisation qui nous donne les primitives de base strictement nécessaires à la factorisation. Le chapitre III contient cet algorithme proche de la stratégie dégagée par l'analyse du chapitre II, les spécifications des primitives de base et des modules-méthodes.

2.A la lumière de la liste des primitives dont on a besoin, nous avons construit, choisit les structures de données en fonction de nos désirs de performance. Le chapitre IV contient cette construction progressive et les raisons des différents choix. Il termine par la déclaration en Pascal des structures de données finales.

Nous disposons alors de tout ce qui est nécessaire à l'implémentation d'un sous-système. Nous avons alors implémenté une des méthodes de factorisation: la mise en évidence. Le chapitre V contient une description générale des fonctions implémentées, l'algorithme de mise en évidence et une description de l'architecture physique. Le code et les spécifications concrètes se trouvent en Annexe 4.

Chapitre II

ANALYSE DE LA FACTORISATION

1.LE NOYAU DU DIDACTICIEL : La factorisation

Une des fonctions primordiales du didacticiel est la factorisation elle-même. Nous devons construire un logiciel qui à partir d'un polynôme (ou expression) quelconque, fournit sa forme factorisée correspondante.

Ce logiciel aura les propriétés de performance habituelles: économie de place mémoire, rapidité d'exécution. Il devra, d'autre part, être construit de telle manière qu'il approche les mécanismes de factorisation que l'homme utilise pour factoriser.

D'ores et déjà, nous pouvons dire que le processus de factorisation est dynamique en ce sens qu'il se compose d'une succession de processus unitaires : ANALYSE- CHOIX- APPLICATION D'UN MECANISME PRECIS. Il faudra que le logiciel offre une souplese d'utilisation c'est à dire , offre des moyens d'action pour l'utilisateur enseignant afin que celui-ci puisse modifier les processus de choix et la succession des différents mécanismes de base.

Maîtriser la factorisation et ses mécanismes sera l'objectif du présent chapitre. Notre analyse portera sur les documents fournis par Monsieur Collinet à savoir,"la définition de la factorisation et exercices typiques" (v.Annexes 1).

2.DEFINITION GENERALE DE LA FACTORISATION

Dans ce paragraphe, nous donnons, d'abord une définition générale de la factorisation. Les définitions précises de chaque concept soulevé sont données dans les paragraphes suivants.

2.1.Les ingrédients de la factorisation

Le mot "factorisation" vient du mot "facteur": il faut faire apparaître le maximum de facteurs dans une expression donnée, il faut transformer des sommes en des produits suivant certaines règles.

A la première lecture de l'Annexe 1, nous avons retiré trois matières premières qui composent essentiellement la factorisation. Il y a, d'abord, le concept de polynômes (ou expression), objets sur lesquels on agit, ensuite, le concept de méthodes de factorisation, actions sur les polynômes, et enfin, la stratégie de factorisation qui englobe l'analyse du polynôme (ou expression) et le choix de la méthode de factorisation à appliquer sur celui-ci.

En réalite, factoriser c'est trouver une suite d'expressions équivalentes. Cette suite est ordonnée de telle manière que chacune de ces expressions s'obtient par application d'une certaine méthode de factorisation sur l'expression précédente et telle que la dernière expression est "factorisée". Il faut trouver un chemin vers la solution, et ce chemin n'est pas nécessairement

le seul possible. En effet, plusieurs combinaisons de méthodes peuvent être applicables sur l'expression initiale.

2.2.Ce qui lie les ingrédients : L'EQUIVALENCE MATHEMATIQUE

Le principe de base sous-jacent à toute transformation (application d'une méthode de factorisation) est l'équivalence mathématique entre l'expression initiale et l'expression résultat. Toute méthode de factorisation transforme une expression exp1 en une expression exp2 identique du point de vue mathématique c'est-a-dire quelles que soient les valeurs de leurs variables, exp1 et exp2 prennent des valeurs numériques égales.

2.3.Les catégories de méthodes

D'une analyse approfondie des exercices typiques (v.Annexe1), nous avons tiré six actions de base applicables sur des polynômes. Nous les avons classées en trois catégories.

La catégorie I de "FACTORISATION PURE" reprend trois méthodes: la MISE EN EVIDENCE, la méthode dite des "FORMES REMARQUABLES" et la méthode dite des "DIVISEURS X-A". Cette catégorie se nomme ainsi car les méthodes qu'elle englobe sont les plus directes pour produire un produit à partir d'une somme, les plus rapides pour atteindre le but fixé.

La catégorie II des "INTERMEDIAIRES A LA FACTORISATION" reprend deux méthodes: la méthode dite de "GROUPEMENT" et la méthode dite d'"ARTIFICES". Le nom de cette dernière est paticulièrement bien choisi car cette catégorie II est celle des méthodes de manipulations des expressions initiales pour faire apparaître des possibilités d'application de méthodes directes de la catégorie I.

La catégorie III d'"AIDE A LA FACTORISATION" regroupe tout ce qu'on entend par EFFECTUER (calculer). Nous raffinerons ce concept d'effectuer plus loin. Cette catégorie aide la factorisation pure. On applique différentes propriétes d'opérateurs et autres règles de calcul sur l'expression initiale pour faire apparaître d'éventuelles possibilités d'application de méthodes pures ou de méthodes intermédiaires.

2.4.L'approche humaine et le logiciel

L'être humain oriente sa démarche de factorisation en analysant la forme de l'expression qu'il a devant les yeux. Donc, sa réussite tient de sa perspicacité, de sa

propention à reconnaître une opportunité d'application d'une méthode de factorisation. De toute facon; il y a des transformations plus difficiles à découvrir que d'autres; il y a des transformations que l'analyse humaine découvre difficilement, ou très rarement.

D'une part, le programme de factorisation devra découvrir les transformations facilement trouvées par l'analyse humaine mais aussi d'autres que l'utilisateur ne trouve habituellement pas.

D'autre part, le programme doit progresser par étapes successives suivant un cheminement proche des mécanismes humains de factorisation. Il doit, donc, s'inspirer aussi de la forme "actuelle" de l'expression à factoriser. En d'autres mots, il ne doit utiliser les transformations "radicales" essentiellement EFFECTUER qu'en dernier ressort car ces méthodes font disparaître de l'information sur l'expression, information qui aurait pu servir à l'application de méthodes de la catégorie I ou II.

3.L'EXPRESSION

3.1.Définition générale

3.1.1.Définition d'un expresion

Nous appelons une EXPRESSION EXP, toute forme mathématique suivante:

$$exp = \sum_{i=1}^{n} \prod_{j=1}^{m_i} exp_{ij}^{e_{ij}}$$

où n, m_i sont >= 1, où e_{ij} est un exposant entier ou rationnel positif, exp_{ij} est une expression ou une valeur ou une variable. Notation:

$\prod$ représente un produit,

$\sum$ représente une somme.

Ce concept d'expression recouvre les différentes formes mathématiques manipulées.

Toute expression est, donc, une somme de produits de puissances (entières positives ou rationnelles positives éventuellement nulles) d'expressions ou (et) de valeurs ou (et) de variables. Elle se base sur les concepts d'opérateurs, de variables, de valeurs.

3.1.2.Les opérateurs

L'opérateur monadique de NEGATION, noté "-". Exemple: -(a*b), -a. Ainsi, la différence entre a et b notée "a-b" est considérée comme la somme de l'expression "a" et de l'expression moins monadique

"-b". En réalié, nous travaillons avec un concept de moins monadique mais nous utilisons une représentation sous forme diadique dans les chaînes de caractères.

L'opérateur diadique d'EXPONENTIATION, noté "^". Exemple: a^3, (a+b)^2, 3^1/2.
Rem. : l'exposant est uniquement entier positif si la base de la puissance est une variable ou une expression contenant au moins une variable. L'exposant est rationnel positif sinon.

L'opérateur diadique d'ADDITION, noté "+". Exemple: a+x*y, a+x*y-2.

L'opérateur diadique de MUTIPLICATION, noté "*". Exemple: a*a*x*12, 1/2*x^2.

3.1.3.Les priorités d'opérateurs

Par ordre décroissant de priorités:
somme +, moins -,
produit *,
puissance ^.

3.1.4.Les valeurs

On désigne les coéficients des expressions sous la dénomination "valeur" par opposition à "variable".
Toute valeur est un entier, un rationnel ou un réel "radical" (2^2/3). Exemple: 2, 245, 242/243, 1/2, 2^1/2.

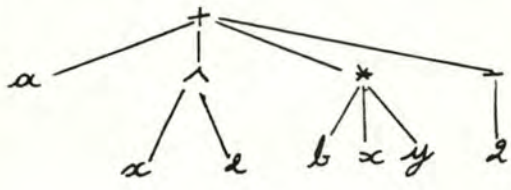
3.1.5.Les variables

Une variable est représentée par une lettre; son domaine est l'ensemble des réels.

3.1.6.La structure d'une expression

Toute expression telle que définie ci-dessus peut se représenter sous la forme d'arbres n-aires.

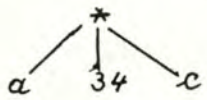
Exemple: a+x^2+b*x*y-2



Représentons chaque type d'opérateur sous forme d'arbre n-aire:

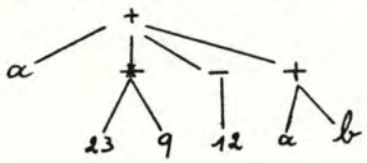
Puissance: a^34

Produit: a*34*c



Moins: -(a*2)

Somme: a+23*q-12+(a+b)



3.2.Expressions équivalentes

L'équivalence mathématique provient de propriétés des opérateurs.

L'assoçiativité de l'addition et de la multiplication:
a+(b+c) = (a+b)+c = a+b+c
a*(b*c) = (a*b)*c = a*b*c où a,b,c sont des expressions.

La commutativité de l'addition et de la multiplication:
a*b = b*a où a, b sont des expressions.
a+b = b+a

Les éléments neutres: 1 pour l'addition, 0 pour la multiplication.
a*1 = 1*a = a
a+0 = 0+a = a où a est une expression.

L'élément absorbant pour la multiplication: 0
a*0 = 0*a = 0 où a est une expression.

La distributivité de la multiplication sur l'addition:
a*(b+c) = a*b+a*c
(a+b)*c = a*c+b*c où a,b,c sont des expressions.

Les propriétés d'exponentiation: x est une expression,
x^1 = x
x^0 = 1, si x <> 0

0^a = 0, où a est un rationnel > 0
x^n * x^m = x^p avec p = m+n, où m,n,p sont des rationnels positifs
(x^n)^m = x^p avec p = n*m, où m,n,p sont des rationnels positifs

Les règles de signes
--x = x
-(a+b+c) = -a-b-c
-(a*b) = a*(-b) = (-a)*b.

3.3.La forme canonique

3.3.1.La forme canonique classique

Soit un polynôme à une seule variable x, à puissances entières, à coéficients réels:

$$P(X) = \sum_{i=0}^{n} a_i * x^{\wedge} i$$

Un tel polynôme est dit sous forme canonique.

3.3.2.La forme canonique "généralisée"

Toute expression exp est sous forme canonique ssi c'est une somme de n termes t (n>0) telle que

chaque terme est

soit une valeur différente de 0 (terme indépendant),

soit une variable,

soit une puissance dont la base est une variable et l'exposant un entier > 1,

soit le produit de m facteurs (m>1) où

-chaque facteur est une valeur (<>1, <>0) ou une variable ou une puissance dont la base est une variable et l'exposant un entier > 1

-toutes les variables sont différentes deux à deux

-il n'y qu'un seul facteur valeur

Il n'y a qu'un seul terme indépendant dans la somme.

Soient la forme canonique somme, $t_1 + \ldots + t_m$ telle que définie ci-dessus, alors pour tout $i_1, i_2, i_1 \neq i_2$ le produit des facteurs variables (ou puissances)

du terme t_{i_1} est différent du produit des facteurs variables (ou puissances) du terme t_{i_2} à la commutativité près. Exemple:
3*x^2*y^3+2*y^3*x^2 n'est pas sous forme canonique car x^2*y^3 est égal à y^3*x^2 à la commutativité près. La forme canonique correspondante est 5*x^2*y^3.

3.4.La forme factorisée

3.4.1.La forme factorisée complète

Si nous prenons un polynôme P(x) à une seule variable x, à puissances entières, à coéficients réels, P(x) est dit factorisé ssi il est de la forme

$$P(x) = \prod_{i=1}^{n} (x - x_i)^{e_i} * R(x)$$

où e_i est un entier positif

x_i est un réel

R(x) ,éventuellement égal à 1, est un produit de polynômes non factorisables c'est à dire n'ayant pas de racines réelles.

$\prod_{i=1}^{n} (x - x_i)^{e_i}$,éventuellement égal à 1, est un produit de puissance (éventuellement égale à un) de polynômes de degré 1 où x_i est une racine réelle du polynôme.

3.4.2.La forme factorisée

Il faut adapter la définition ci-dessus car d'une part, quelle est la forme finale de R(x)? Supposons que R(x) soit un polynôme de degré 4 non factorisable (comme ci-dessus), apparaîtra-t-il sous sa forme degré 4 ou sous la forme d'un produit de deux polynômes de degré 2 non factorisables?
D'autre part, il faut pouvoir découvrir les racines. Au delà du degré 5, en général, il est rare de trouver toutes les racines. Factoriser complètement est en général inaccessible.
Il faut tenir compte des moyens (méthodes) dont on dispose pour factoriser. A partir d'un énoncé expi, on essaie de réduire le degré des composants, de faire apparaître le maximum de facteurs grâce à des méthodes bien précises. Dans cet esprit, une forme est dite FACTORISABLE si il est possible de lui appliquer une méthode de factorisation de la catégorie I et II. Si expi+1 est l'expression résultat de la factorisation de expi alors expi+1 est dite factorisée: chacun de ses facteurs est tel

qu'il est impossible de lui appliquer une méthode de factorisation (catégorie I ou II).

4.LES METHODES DE FACTORISATION

Toute méthode de factorisation transforme une expression expi en une autre expression expi+1 équivalente.

4.1.La mise en évidence

La mise en évidence est une application directe de la distributivité de la multiplication sur l'addition.
En terme d'expression, on peut définir la mise en évidence comme suit: si expi est une expression somme de n termes (1 < n) de la forme:

$\sum_{k=1}^{n} (q * r_k)$ alors l'expression équivalente expi+1, résultat de l'application de la mise en évidence sur expi est un produit de la forme:

$q * (\sum_{k=1}^{n} r_k)$ où q et r_k sont des expressions.

Exemple: (x-2)^2*y^3+y^2*(x-2)^3+(x-2)*y^2 =
(x-2)*y^2*((x-2)*y+(x-2)^2+1)

4.2.Les formes remarquables

La méthode des formes remarquables consiste en l'application de mécanismes représentés par les formules suivantes:

Soient a,b,c des expressions,

la différence de deux carrés a^2-b^2 = (a+b)*(a-b);
la différence de deux cubes a^3-b^3 = (a+b)*(a^2*+a*b+b^2);.
la somme de deux cubes a^3+b^3 = (a^2-a*b+b^2);
le trinôme carré parfait a^2+2*a*b+b^2 = (a+b)^2 et a^2-2*a*b+b^2 = (a-b)^2
autre carré parfait a^2+b^2+c^2+2*a*b+2*a*c+2*b*c = (a+b+c)^2

le trinôme du second degré: utilisation du réalisant ou décomposition du terme indépendant (v.Annexe 1);

Les quotients remarquables x^m-a^m divisible par x-a, x^m+a^m est divisible par x+a si m est impair, x^m-a^m est divisible par x+a si m est pair.

4.3.La méthode des diviseurs X-A

Soit une expression de la forme
expi = $\sum_{i=1}^{n} a_i * x^{\wedge} i$ avec a_i coéficients entiers, x une expression. Si toutes les racines du polynômes expi sont entières alors le procédé suivant les découvre toutes sinon il peut en découvrir: sélectionner les diviseurs entiers, a, du terme indépendant; calculer $\sum_{i=1}^{n} a_i * a^{\wedge} i$; sélectionner les a qui annulent l'expression. Appliquer l'algorithme suivant:

res := 1; expi+1 := expi;

pour chaque racine entière, a, faire: expi+1 := quotient de la division euclidienne (ou division de Horner v.Annexe 1) de expi+1 par x-a.

Res := res * (x-a)

expi+1 := res * expi+1.

4.4.Le groupement

Appartenant à la catégorie des intermédiaires à la factorisation, cette méthode consiste à appliquer directement la propriété de commutativité de l'addition pour faire apparaître une ou plusieurs possibilités d'application de méthodes de factorisation pure. On forme n groupes (n>= 2) d termes (un groupe est une somme de termes de l'expression initiale). Chaque groupe (ou au moins un) subit une ou plusieurs applications de méthodes de factorisation pure (le plus souvent mise en évidence et formes remarquables). Les résultats replacés dans la somme initiale forment une expression équivalente dont la forme suggère l'application d'une méthode de factorisation pure sur l'ensemble.

Le choix des termes d'un groupe ne se fait pas au hasard, il est dicté par le succès final de la dernière étape: transformer l'ensemble.

Exemple:

```
x^4-2*x^3+x-2 = (x^4-2*x^3)+(x-2)
                grouper
              = x^3*(x-2)+(x-2)
                m-é-é sur le groupe 1
              = (x-2)*(x^3+1)
                m-é-é sur l'ensemble
              = (x-2)*(x+1)*(x^2-x+1)
                somme de deux cubes
```

4.5.L'artifice

Cette méthode, intermédiaire à la factorisation pure, consiste à manipuler l'expression initiale (une somme) pour faire apparaître une possibilité d'application de

méthodes de factorisation pure.

La première manipulation consiste à préparer l'expression à un groupement. On procède soit à un éclatement (exemple: 5*x = 3*x+2*x) soit à un ajout-retrait d'une valeur (+3-3). Dans les deux cas, il faut choisir judicieusement les valeurs à éclater ou à ajouter-retirer. L'éclatement utilise un terme déjà existant, par contre, on ajoute-retire un nouveau terme (valeur) (exemple: expi = 3*x^2+5*x; expi+1 = 3*x^2+5*x+1-1, on a ajouté-retiré le terme "1").

La seconde manipulation: appliquer un groupement de termes judicieux (le choix de ce groupement est influencé par le choix de l'éclatement ou du terme ajouté-retiré).

La troisième manipulation comprend l'application sur chaque groupe (ou au moins un groupe) d'une ou plusieurs méthodes de factorisation.

La quatrième manipulation: appliquer une méthode de factorisation (essentiellement, la mise en évidence et les formes remarquables) sur l'ensemble de l'expression obtenue par la somme des résultats issus de la troisième manipulation. Il faut que le choix des valeurs d'éclatement ou d'ajout-retrait et le choix du groupement soient tels que la quatrième manipulation soit fructueuse. On sent bien qu'il y a dans l'artifice plus de "feeling" que dans les autres méthodes.

Illustrons, d'abord, la démarche générale sur un exemple:

expi x^3+2*x-3

1. éclatement: 2*x = -x+3*x
   x^3-x+3*x-3
2. groupement: gr1: x^3-x
   gr2: 3*x-3
3. fact. pure: gr1: m-é-é x*(x^2-1)
   f-rem x*(x-2)*(x+2)

   gr2: m-é-é 3*(x-1)
4. ensemble: x*(x-2)*(x+2)+3*(x-1)
   m-é-é : (x-1)*(x*(x+1)+3)

Précisons la démarche:

4.5.1.L'ajout-retrait

soit expi $= \sum_{i=1}^{m} a_i * x^{\wedge} i$ où i entier, x est une expression, a une valeur. Sélectionner les diviseurs entiers, d du terme indépendant comme dans la méthode des diviseurs X-A.

Pour chaque puissance n (sauf la plus basse) de x dans expi, choisir soit l'ajout (x^n-d) soit le retrait (x^n+d) ce qui permettra l'application d'une forme remarquable sur le groupement x^n+d ou x^n-d.

Calculer (effectuer) la somme du terme indépendant et des différentes valeurs retirées ou ajoutées; ceci donne une valeur v que l'on essaie de grouper

avec la puissance (>= 1) de x la plus basse.
L'ajout-retrait réussit s'il est possible d'appliquer la mise en évidence ou une des formes remarquables sur l'expression générale (reprennant les différents résultats issus de chaque groupe). On procède par essais successifs de chaque diviseur d du terme indépendant.

Exemple: x^3+2*x-3
Diviseurs: 1,-1,3,-3
Essai d: 1
1.ajout-retrait: x^3-1+2*x-3+1
2.grouper: gr1: x^3-1
gr2: 2*x-3+1
2*x-2 en effectuant (v)
3.appliquer fact. pure sur chaque groupe:
gr1: (x-1)*(x^2+2*x+1) par f-rem
gr2: 2*(x-1) par m-é-é
4.appliquer fact. pure sur l'ensemble:
(x-1)*((x^2+2*x+1)+2) par m-é-é

Exemple d'échec: x^3+4*x+5
Diviseurs: 1,5,-1,-5
Essai d: 1
1.ajout-retrait: x^3-1+4*x+5+1
2.grouper: gr1: x^3-1
gr2: 4*x+5+1
4*x+6 en effectuant
3.appliquer fact. pure sur chaque groupe:
gr1: (x-1)*(x^2+2*x+1) par m-é-é
gr2: 2*(x+3) par m-é-é
4.Il est impossible d'appliquer une méthode de factorisation pure sur l'expression générale: (x-1)*(x^2+2*x+1)+2*(x+3)

4.5.2.L'éclatement

Pour une somme à trois termes,
a*x^m+b*x^n+c où a, b, c sont des valeurs, m>n, des entiers, x une expression. Sélectionner les diviseurs entiers du terme indépendant. Choisir d parmi eux. Prendre le terme à puissance n >= 0 la plus basse: b*x^n. Eclater ce terme: a*x^m-e*x^n+d*x^n+c tel que -e+d = b. Grouper: a*x^m-e*x^n et d*x^n+c. Suivre la procédure déjà décrite pour l'ajout-retrait. L'éclatement réussit si une méthode (m-é-é ou f-rem) est applicable sur au moins un des deux groupes et si une méthode de factorisation est applicable sur l'expression générale.

Exemple: x^3+2*x-3
Diviseurs: 1,-1,2,-2,3,-3

```
choix: 3
1.étlat.: x^3+3*x-x-3
2.grouper: gr1: x^3-x
           gr2: 3*x-3
3.fact. pure sur chaque groupe:
  gr1: x*(x^2-1) par m-é-é
       x*(x-1)*(x+1) par f-rem
  gr2: 3*(x-1) par m-é-é
4.fact. pure sur l'expression générale:
  x*(x-1)*(x+1)+3*(x-1) = (x-1)*(x*(x+1)+3)
par m-é-é.
```

Autre exemple: a^8+a^4-2

```
= a^8+a^4+2*a^4-2
= a^4*(a^2-1)+2*(a^4-1)
= a^4*(a+1)(a-1)+2*(a^2-1)*(a^2+1)
= a^4*(a+1)*(a-1)+2*(a+1)*(a-1)*(a^2+1)
= (a+1)*(a-1)*(a^4+2*(a^2+1)).
```

4.6.Effectuer

Cette méthode est une aide à la factorisation qui, en aucun cas, ne transforme une somme en un produit. Il faut distinguer "EFFECTUER AU 1er DEGRE" ET "DEPARENTHISER".

4.6.1.Effectuer au 1er degré

Cette méthode regroupe des "calculs" dont voici quelques exemples:

```
1+4=5
a^n*a^m=a^(n+m)
2^2=4
(a^n)^m= a^(n*m)
3*a+7*a=10*a
3*4=12
3*a*4=12*a
```

Cette méthode règle les problèmes de signes (-1)*(-1)=1, -(-1)=1, élimine les parenthèses inutiles: ((a+b))^2 devient (a+b)^2. Une paire de parenthèses est inutile si elle est en double avec une autre paire. Exemple:

((a+b)) comprend une paire de parenthèses inutiles, a*(a+b) n'a pas de parenthèse inutile, a*((a+b)) a une paire de parenthèses inutiles.

Effectuer au 1er degré, c'est obtenir une expression de la forme:

$$\sum_{i=1}^{m} a_i \prod_{j=1}^{n_i} exp_{ij}\,\hat{}\,e_{ij}$$

où exp_{ij} différent d'une valeur

a_i valeur non nulle

e_{ij} exposant non nul

$exp_{ij_1} \neq exp_{ij_2}$, $\forall j_1 \neq j_2$

exp_{ij} effectué au 1er degré , $\forall i, j$

$\prod_{j=1}^{n_{i_1}} exp_{ij}{}^{\wedge e_{ij}}$ est différent de $\prod_{j=1}^{n_{i_2}} exp_{ij}{}^{\wedge e_{ij}}$ $\forall i_1 \neq i_2$,

à la commutativité près;

on a appliqué les règles de signes (v. § 3.2);

les parenthèses inutiles sont éliminées.

Si exp est effectuée au 1er degré, tous ses termes (si c'est une somme), tous ses facteurs (si c'est un produit), sa base et son exposant (si c'est une puissance) sont effectués au 1er degré.

4.6.2.Déparenthiser

Consiste à enlever un niveau de parenthèses c'est-à-dire:

1.effectuer au 1er degré

2.appliquer la distributivité (dans le "sens" inverse de la mise en évidence)

3.appliquer les formules remarquables dans le "sens" opposé de la méthode "formes remarquables". Exemple: (a-b)*(a+b) devient a^2-b^2.

Attention, on n'enlève qu'un seul niveau de parenthèses à la fois. Ainsi,
a*((b+1)*(b-1)+c)+3*(4+x) devient
a*(b+1)*(b-1)+a*c+12+3*x et non pas
a*(b^2-1)+a*c+12+3*x.

5.CONCLUSION: LA STRATEGIE

5.1.La démarche humaine

Factoriser expi veut dire trouver l'expression factorisée équivalente à expi en un certain nombre d'étapes représentant chacune l'application d'une méthode de factorisation.
Ce dynamisme, cette approche progressive de la forme factorisée est orienté par l'objectif premier: faire

apparaître un maximum de facteurs. A chaque étape, l'homme analyse l'expression courante, déduit la méthode qu'il va essayer d'appliquer. Celle-ci échoue ou réussit. Dans le premier cas, l'homme va essayer autre chose. S'il a épuisé toutes les méthodes qu'il connaît ou s'il a épuisé sa perspicacité, il déclare cette expression courante comme factorisée. Dans le second cas, il reprend le processus d'analyse-choix-essais jusqu'à la forme factorisée.

5.2.La notion de priorité

Le choix d'une méthode est inspiré par la forme courante qu'on a devant les yeux. Mais il y a des priorités dans le choix. Habituellement, on choisit d'abord la mise en évidence, ensuite les formes remarquables car elles transforment directement une somme en un produit sans calcul compliqué. On choisit, ensuite, les méthodes de la catégorie II qui demandent plus de "feeling", plus de calculs. La plupart du temps, on effectue en dernier lieu.

5.3.Forme canonique et effectuer

Toute expression a une expression canonique équivalente. Celle-ci s'obtient par calculs successifs, par applications successives de déparenthiser jusqu'à ce que cette méthode échoue. Deux expressions sont équivalentes si elles ont même expression canonique.

5.4.Les niveaux de canonisation

On pourrait imaginer un procédé systématique de factorisation: déparenthiser l'expression initiale jusqu'à la forme canonique. Ensuite appliquer la méthode des diviseurs X-A. Cette méthode demande beaucoup de calculs. De plus, elle ne donne pas toutes les racines entières dans tous les cas.

En réalité, il existe des niveaux de canonisation de l'expression initiale. Chaque niveau correspond à l'application d'une fois déparenthiser. Chaque déparenthisation nous rapproche de la forme canonique, nous éloigne de l'expression initiale. Les méthodes comme artifice et groupement, mise en évidence et formes remarquables, utilisent les informations que les méthodes comme effectuer font disparaître. Déparenthiser fait disparaître des parenthèses, des groupements. Déparenthiser distribue (fait disparaître des facteurs), la mise en évidence fait le contraire. Effectuer au 1er degré transforme -2+3 en 1 alors que l'artifice peut éclater 1 en -2+3.

C'est pourquoi on descend d'un niveau de parenthèses en dernier lieu. On essaie d'utiliser au maximum l'information qui existe dans l'expression courante pour les méthodes de factorisation pure.

Chapitre III

CONSTRUCTION D'UN ALGORITHME DE FACTORISATION

1.MOTIVATIONS

Les mécanismes de base de la factorisation étant bien définis, il nous faut trouver des structures de données représentant les expressions et répondant aux critères suivants:

1.Rapidité d'actions de leurs primitives de base permettant un temps de réponse de factorisation raisonnable.

2.Place mémoire occupée raisonnable

3.Simplicité des algorithmes de factorisation.

L'objet du présent chapitre est de préciser les primitives strictement nécessaires à la réalisation d'un algorithme de factorisation. La démarche de construction de l'algorithme est top-down. Le langage est un simili- Pascal, les spécifications en pré-, post-conditions.

2.STRATEGIE DE FACTORISATION

Cet algorithme va séquencer les différentes méthodes de factorisation suivant une stratégie proche de celle découverte au chapitre II. Le schéma de factorisation est le suivant:

Soit expd une expression à factoriser. Chaque méthode est un module avec l'expression à transformer en entrée; en sortie: une réponse positive et le résultat si la méthode réussit, une réponse négative sinon.

Séquencement des méthodes:

1.effectuer-1er-degré

2.filtrer l'expression expd c'est à dire:
Si expd est une puissance alors
la solution est la base factorisée à la même puissance.
Si expd est un produit alors
si expd est un terme de base
(terme de base = produit de facteurs de base)
alors expd est factorisée
sinon
la solution est le produit des facteurs de expd factorisés.
Si expd est un moins alors
la solution est la négation de la "base" factorisée.
Si expd est une variable ou une valeur alors
la factorisation échoue
Si expd est une somme d'au moins deux termes alors
on essaie de lui appliquer les méthodes de factorisation suivant certaines priorités.

3.Essais successifs des méthodes:
3.1.D'abord essayer les méthodes de la

catégorie I, ensuite, de la catégorie II. Dès qu'une méthode réussit (expd transformée en exp1), expd est considérée comme factorisable et on essaie de factoriser chaque facteur de exp1.

3.2.Si aucune de ces méthodes ne réussit, on déparenthise expd en exp1. La solution est exp1 factorisé.

3.3.Si il n'y a plus moyen de déparenthiser, expd n'est pas factorisable.

Rem.: Définition d'un facteur de base
C'est une expression valeur ou variable, une expression puissance dont la base est une variable ou une valeur, une expression moins dont la "base" est une valeur ou une variable.

3.L'ALGORITME EN SIMILI-PASCAL

```
Function FACTORISER (exp : expression,var expf : expression)
: boolean

  Prcd:
    exp et expd sont des expressions; expf est une somme
    d'au moins deux termes.

  Pscd:
    Renvoie vrai
      si expd est factorisable (au moins, une méthode des
      catégories I et II a réussi) auquel cas expf est le
      résultat d'applications successives de méthodes de
      factorisation suivant le schéma ci-dessus.
      équi-math(expd,expf).
      Pour tout facteur f de expf, factoriser (f,f1) =
      false.
      Si expf est une puissance, factoriser (base,f1) =
      false.
      Si expf est un moins (-expi), factoriser (expi) =
      false.
      Si expf est une somme, factoriser (expf) = false.
      Dans les autres cas, factoriser (expf) = false.

    renvoie faux
      si expd n'est pas factorisable auquel cas expf =
      l'expression triviale "1" notée UN.

Rem.: Equi-math (exp1,exp2) signifie que exp1 et exp2 sont
des expressions équivalentes.

begin
  exp0 := expd; expf := un;
  exp0 := effectuer-1er-degré (exp0);
  if produit (exp0) then
```

```
  begin
    if terme-base(exp0) then factoriser := false
    else
      begin
        while get-fact(exp0,f) do
          if factoriser (f,f1) then expf := (expf,f1)
      end
  end
else
  begin
    e := exposant (exp0);
    if e <> 1 then
      begin
        expb := base (exp0);
        if terme-base (expb) then factoriser := false
        else
          if factoriser (expb,expi) then
            expf := prod (expf,expos (expi,e))
          else factoriser := false            end
    else
      if facteur-base (exp0)
        then factoriser := false
      else
      begin (* exp0 est une somme *)
        fini := false;
        while (not fini) do
          begin
            if essai-fact (exp0,exp1) then
              (* une des méthodes de factorisation
                 a réussi *)
              begin
                fini := true;
                factoriser := true;
                while get-fact (exp1,f) do
                  if factoriser (f,f1) then
                    expf := prod (expf,f1)
              end
            else
              begin
                (* aucune méthode de factorisation
                   n'a réussi*)
                (* on essaie de déparenthiser *)
                expi := déparenthiser (exp0);
                if expi = un then
                  (* déparenthiser a échoué *)
                  begin
                    fini := true;
                    factoriser := false
                  end
                else
                  (* on essaie de factoriser
                    exp0 déparentise *)
                  exp0 := expi
              end
      end
```

```
      end
    expf := effectuer-1er-degré (expf)
  end;
```

4.MODULES-METHODES ET PRIMITIVES

4.1.Le module essais-fact

Function ESSAI-FACT (expd: expression, var expf: expression): boolean;

Prcd:
expd et expf sont des expressions. Expd est une sommme d'au moins deux termes.

Pscd:
renvoie vrai
si une des méthodes de factorisaion a réussi sur expd (mise en évidence, forme remarquables, groupement, artifice, diviseur X-A) auquel cas expf en est le résultat.
Equi-math (expd,expf)
renvoie faux
sinon auquel cas expf = un

4.2.Les modules-méthodes

4.2.1.Effectuer-1er-degré

Function EFFECTUER-1ER-DEGRE (expd: expression) : expression;

Prcd: exp est une expression

Pscd: renvoie l'expression obtenue en appliquant les transformations décrites au § 4.6.1. du chapitre II.

4.2.2.M-é-é

Function M-E-E (expd: expression; var expf: expression): boolean;

Prcd: expd, expf sont des expressions.
Expd est une somme d'au moins deux termes.

Pscd: renvoie vrai si la mise en évidence telle que décrite dans le § 4.1. du chapitre II, a réussi sur expd auquel cas expf en est le résultat; renvoie faux sinon, auquel cas expf = un.

4.2.3.F-rem

Function F-REM (expd: expression; var expf: expression): boolean;

Prcd: expd et expf sont des expressions. Expd est une somme d'au moins deux termes.

Pscd: renvoie vrai si la méthode des formes remarquables, telle que décrite au § 4.2. du chapitre II, a réussi sur expd auquel cas expf en est le résulat; renvoie faux sinon, auquel cas expf = un.

4.2.4.Grouper

Function GROUPER (expd : expression; var expf: expression): boolean;

Prcd: expd et expf sont des expressions. Expf est une somme d'au moins deux termes.

Pscd: renvoie vrai si la méthode des groupements, telle que décrite au § 4.4. du chapitre II, a réussi, auquel cas expf en est le résultat; renvoie faux sinon, auquel cas expf vaut un.

4.2.5.Artifice

Function ARTIFICE (expd: expression; var expf: expression): boolean;

Prcd: expd et expf sont des expressions. Expd est une somme d'au moins deux termes.

Pscd: renvoie vrai si la méthode des artifices, telle que décrite dans le § 4.5. du chapitre II, a réussi sur expd, auquel cas expf en est le résultat; renvoie faux sinon, auquel cas expf = un.

4.2.6.Déparenthiser

Function DEPARENTHISER (exp: expression): expression;

Prcd: exp est une expression

Pscd: renvoie un si la méthode de déparenthisation décrite au § 4.6.2. du chapitre

II n'a pas réussi sur exp; renvoie le résultat de la déparenthisation sinon.

4.2.7.Diviseur X-A

Function DIVISEURX-A (exp: expression; var expf: expression): boolean;

Prcd: expd et expf sont des expressions. Expd est une somme d'au moins deux facteurs.

Pscd: renvoie vrai si la méthode des diviseurs X-A, telle que décrite au chapitre II § 4.3., a réussit auquel cas expf en est le résultat; renvoie faux sinon, auquel cas expf vaut un.

4.3.Les primitives

4.3.1.La primitive d'égalité d'expressions

Function EGAL-CH (exp1, exp2: expression): boolean;

Prcd: exp1 et exp2 sont des expressions.

Pscd: renvoie vrai si exp1 et exp2 sont représentées par la même chaîne de caractères; renvoie faux sinon.

4.3.2.Les primitives de construction d'expressions

Function PROD (exp1, exp2: expression): expression;

Prcd: exp1 et exp2 sont des expressions.

Pscd: renvoie l'expression produit de exp1 et de exp2.

Function SOM (exp1,exp2: expression): expression;

Prcd: exp1 et exp2 sont des expressions.

Pscd: renvoie l'expression somme de exp1 et exp2.

Function EXPOS (exp1: expression; exp2: exposant): expression;

Prcd: exp1 est une expression; exp2 est un exposant c'est-à-dire un rationnel positif si exp1 est une valeur, un entier positif sinon.

Pscd: renvoie l'expression puissance de base exp1

et d'exposant exp2.

Function MOINS (exp: expression): expression;

Prcd: exp est une expression.

Pscd: renvoie l'expression négation de exp.

Function QUOT (expd, expf: expression): expression;

Prcd: expd et expf sont des expressions. Expd est une somme d'au moins deux termes. Expf est un facteur commun à chaque terme de expd.

Pscd: expd est l'expression résultat de la mise en évidence du facteur commun expf dans expd.
Equi-math(expd,prod(expf,quot(expd,expf))).

4.3.3.Les primitives d'extraction de sous-expressions

Function BASE (exp: expression): expression;

Prcd: exp est une expression.

Pscd: renvoie la base de exp si exp est une puissance; renvoie exp si ce n'est pas une puissance.

Function GET-FACT (exp1,exp2: expression): boolean;

Prcd: exp1 et exp2 sont des expressions.

Pscd: renvoie vrai si exp1 <> un, auquel cas exp1 est amputé de son premier facteur (si après amputation exp1 n'a plus de facteur, alors exp1 = un); exp2 est ce premier facteur;
renvoie faux sinon, auquel cas exp1 et exp2 valent un.

Function GET-TERME (exp1,exp2): boolean;

Prcd: exp1 et exp2 sont des expressions.

Pscd: renvoie vrai si exp1 <> zéro, auquel cas exp1 est amputé de son 1er terme ; exp1 vaut zéro si il n'y avait qu'un terme; exp2 vaut ce premier terme; renvoie faux sinon auquel cas exp1 et exp2 valent zéro.

Function EXPOSANT (exp: expression): type-exposant;

Prcd: exp est une expression.

Pscd: renvoie la puissance de exp; si celle-ci n'est pas une puissance, renvoie 1.

4.3.4.Les primitives d'extraction de "renseignements"

Function PRODUIT (exp: expression): boolean;

Prcd: exp est une expression.

Pscd: renvoie vrai si exp est un produit; faux sinon.

Function SOMME (exp: expression): boolean;

Prcd: exp est une expression.

Pscd: renvoie vrai si exp est une somme; faux sinon.

Function VALEUR (exp: expression): type-valeur;

Prcd: exp est une expression.

Pscd: renvoie vrai si exp est une valeur; renvoie faux sinon.

On devine, aisément, les spécifications des fonctions suivantes: PUISSANCE, VARIABLE, MOINS.

Function TERME-BASE: (exp: expression): boolean;

Prcd: exp est une expression.

Pscd: renvoie vrai si exp est une facteur de base ou un produit de facteurs de base; renvoie faux sinon.

Function FACTEUR-BASE (exp : expression): boolean;

Prcd: exp est une expression.

Pscd: renvoie vrai si exp est un facteur de base (v. definition § 3. du présent chapitre); renvoie faux sinon.

Function FACT-COM (expd,expf: expression): boolean;

Prcd: expd et expf sont des expressions. Expd est une somme d'au moins deux termes.

Pscd: renvoie vrai si expf est un facteur commun à tous les termes de expd; renvoie faux sinon.

Soient expd = $t_1 + \ldots + t_m$ $(m \geq 2)$

où $t_i = f_{i1} * \ldots * f_{im_i}$ $(m_i \geq 1)$

$f_{ij} = f'_{ij} \wedge e_{ij}$ $(e_{ij} \geq 1)$

$f = f' \wedge e$ est un facteur commun aux termes de expd ssi

$\forall 1 \leq i \leq m$, il existe $1 \leq k_i \leq m_i$

tel que $f' = f'_{ik_i}$

et $1 \leq e \leq e_{ik_i}$.

4.3.5.Les primitives liées à l'équivalence d'expressions

Function CANONIQUE (exp: expression): expression;

Prcd: exp est une expression.

Pscd: renvoie l'unique expression canonique (v. § 5.2. chapitre II) associée à exp.

Function EQUI-MATH (exp1, exp2: expression): boolean;

Prcd: exp1, exp2 sont des expressions .

Pscd: renvoie vrai si exp1 et exp2 ont même forme canonique associée.

Chapitre IV

CONSTRUCTION DES STRUCTURES DE DONNEES

1.MOTIVATIONS

Par le chapitre précédent, nous disposons des primitives de base nécessaires à la construction d'un algorithme de factorisation. A la lumière de leurs spécifications, l'objet de ce chapitre est de bâtir les structures de données permettant un temps de réponse optimal, en occupant le minimum de place mémoire et en simplifiant au maximum l'implémentation de ces primitives.

2.LA VIE D'UNE EXPRESSION

L'être humain travaille sur des symboles, sur une représentation des expressions sous forme de chaînes de caractères. C'est sous cette forme qu'une expression "entrera" ou "sortira" dans la machine. Toute expression provenant d'un utilisateur pour l'ordinateur, toute expression destinée à l'utilisateur sera représentée sous forme de chaînes de caractères ayant une syntaxe propre aux expressions manipulées.

Le programme travaille sur une représentation adaptée aux besoins de performance. Il devra transformer la chaîne de caractères en la représentation interne équivalente.

Ensuite, le programme construit une représentation de la solution à la tentative de factorisation sur la représentation interne.

En dernier lieu, il transforme cette solution interne en une suite de chaînes de caractères destinée à l'extérieur.

3.CONSTRUCTION PROGRESSIVE DE LA REPRESENTATION INTERNE

En général, il ya deux choses à représenter: la structure de l'expression et les "renseignements" sur l'expression (v. extraction de "renseignements" dans le § 4.3.4. du chapitre III).

3.1.Représentation sous forme d'arbre

Toute expression a une structure intrinsèque d'arbre n-aires (v.chapitre II § 3.1.1.). Nous avons décidé d'utiliser un arbre binaire spécial: l'ARBRE FILS-FRERE. Celui-ci reflète bien l'utilisation qu'on veut en faire notamment pour les parcours des fils d'un même père.

Définition de l'arbre fils-frère: chaque noeud comprend son type et deux références à d'autres noeuds; une référence à son premier fils, une autre à son frère. Chaque dernier fils référencera son père par son champs "référence frère". Un noeud racine d'une expression n'a pas de frère. Ainsi, chaque noeud comprend un nombre fixe de références, ce qui semble simplifier l'implémentation de l'arbre.

Par type de noeud:

Soient père, fils, frère, f1, f2, f3 des références à

des noeuds, "pas" signifie qu'il n'y a pas de référence au fils, au frère suivant le cas.

```
noeud somme: a+b+c
  père:      type:                somme
             référence fils: f1
             référence frère:?

  f1: type:              variable a
      référence fils: pas
      référence frère:f2

  f2:  type:              variable b
       référence fils: pas
       référence frère:f3

  f3:  type:              variable c
       référence fils: pas
       référence frère:père


noeud moins: -a
      père: type: moins
            référence fils: fils
            référence frère:?

      fils: type: variable a
            référence fils: pas
            référence frère:père

noeud valeur: 3
      père: type: valeur 3
            référence fils: pas
            référence frère:?

noeud variable: a
      père: type: variable a
            référence fils: pas
            référence frère:?

noeud produit: a*b*c
      père: type: produit
            référence fils: f1
            référence frère:?

      f1: type: variable a
          référence fils: pas
          référence frère:f2

      f2:  type: variable b
           référence fils: pas
           référence frère:f3

      f3: type: variable c
          référence fils: pas
```

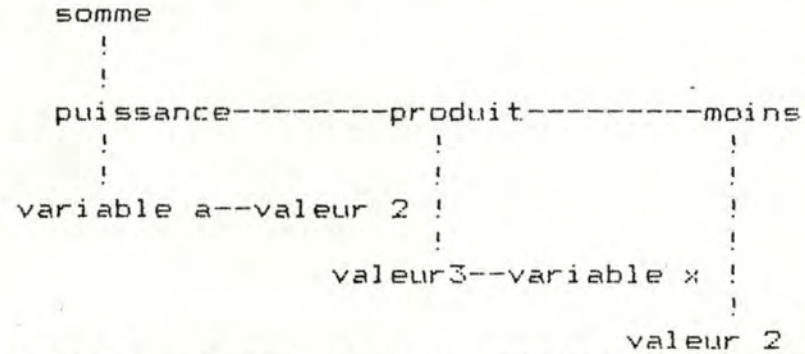
```
                référence frère:père

    noeud puissance: a^2
      père: type: puissance
            référence fils:  fils
            référence frère: ?

      fils: type: variable a
            référence fils: pas
            référence frère: frère

      frère:type: valeur 2
              référence fils: pas
              référence frère: père
```

Exemple: a^2+3*x-2



Les traits horizontaux représentent les références "frère", les traits verticaux les références "fils". Pour que la figure ci-dessus soit complète, il faut rajouter la référence au père pour chaque dernier fils.

3.2.Les sous-expressions et les types d'expressions

Les sous-expressions (termes d'une somme, facteurs d'un produit, base et exposant d'une puissance, "base" d'un moins) apparaissent clairement et sont accessibles dans la structure d'arbre fils-frère. Il y va de même pour les types de noeuds (somme, produit, moins, puissance, valeur, variable).

3.3.Le champs "chaîne de caractères"

L'être humain travaille sur des chaîne de caractères. Un des mécanismes de base qu'il utilise très souvent dans l'analyse d'une expression est d'établir l'égalité entre deux chaînes de caractères représentant chacune une expression. Ceci doit transparaître dans la représentation interne.

Si nous gardons l'arbre fils-frère tel quel, il nous semble que étalir l'égalité de chaînes demande des parcours d'arbre relativement compliqué et long.

C'est pourquoi nous ajoutons un champs à chaque noeud de l'arbre fils-frère initial. Chaque noeud N comprendra un champs "chcar" qui contiendra la représentation sous forme de chaînes de caractères de l'expression dont N est la racine.

Exemple: a*b+a^2

```
  père: chcar: a*b+a^2
        type : somme
        reffi: f1
        reffr: pas de frère

  f1  : chcar: a*b
        type : produit
        reffi: f11
        reffr: f2

  f2  : chcar: a^2
        type : puissance
        reffi: f21
        reffr: père

  f11 : chcar: a
        type : variable
        reffi: pas de fils
        reffr: f12

  f12 : chcar: b
        type : variable
        reffi: pas de fils
        reffr: f1

  f21 : chcar: a
        type : variable
        reffi: pas de fils
        reffr: f22

  f22 : chcar: 2
        type : valeur
        reffi: pas de fils
        reffr: f2
```

où reffi signifie "référence fils", reffr "référence frère", père et les "f" représentent des références à des noeuds.

Cette décision d'introduire chcar dans chaque noeud apporte une fameuse redondance avec la structure du sous-arbre. Cela risque de nous gêner pour la place mémoire. Mais, d'autre part, nous gagnons beaucoup de temps lors de la comparaison des expressions (point de vue chaîne de caractères) et lors de l'extraction de sous-chaîne hors de la représentation interne.

3.4.Ordonnancement de l'arbre fils- frère

Pour factoriser, il est nécessaire de comparer des sous-arbres, des sous-expressions (exemple: pour les facteurs communs aux termes d'une somme). Il faut pouvoir établir l'égalité de chaînes à la commutativité près (égalité de formes canoniques).

3.4.1.Définition

Pour éviter de tenir compte de toutes les permutations possibles des opérandes d'un produit ou d'une somme, nous allons ordonner les fils d'un même père dans l'arbre fils-frère. L'ordre total sur les type des opérandes se définit comme suit: ordre (somme) < ordre (produit) <ordre (puissance) < ordre (variable) < ordre (valeur) < ordre (moins).

Exemple: x^2-3+a*b devient a*b+x^2-3.

L'ordre sur les opérandes de même type se défini comme suit: pour les variables, l'ordre alphabétique; pour les valeurs, l'ordre numérique. L'ordre pour les opérandes d'une puissance: ordre (base) < ordre (exposant).

3.4.2.L'ordre initial

L'utilisateur entre une expression dont les opérandes sont dans un certain ordre. Après les constructions de l'arbre fils-frère ordonnancé, cet ordre initial risque d'être bouleversé. Or, l'utilisateur tient à retrouver dans l'expression résultat un ordre proche de celui qu'il a entré. Notamment lors d'une mise en évidence, si l'utilisateur a entré l'expression a*c+a*b, il attends le résultat a*(c+b) qui respecte l'ordre initial et non le résultat a*(b+c) qui respecte l'ordre interne.

C'est pourquoi chaque noeud aura un champs "ordre initial" qui contiendra le numéro d'ordre de l'opérande dans l'expression initiale. Ce champs est initialisé lors de la construction de l'arbre fils-frère. Chaque transformation de l'expression initiale devra veiller à conserver ce champs. Le programme sortira les opérandes dans l'ordre indiqué par les champs "ordre initial".

Exemple:

expression initiale:	a*c+a*b
ordonnancement:	a*b+a*c
mise en évidence:	a*(b+c)
sortie:	a*(c+b)

3.4.3.Variation de longueur significative

Une chaîne de caractères est définie par sa longueur et par les symboles qu'elle contient dans un certain ordre. Lorsqu'on ordonne une expression, il y a un cas particulier où la chaîne de caractères résultat perd de sa longueur. Illustrons ce cas par un exemple: -a+b devient b-a. On pourrait admettre ce fait mais alors, lorsque on revient à l'ordre initial, il y a une augmentation de longueur (b-a => -a+b). Pour éviter une gestion fastidieuse de variation de longueur dans les chaînes de caractères, on permet l'introduction du caractère "blanc". Si '!'représente le caractère blanc, -a+b devient !b-a. On dira qu'il y a diminution de longueur significative.

3.5.Le champs "effectué au 1er degré"

Effectuer au 1er degré est une transformation qui regroupe beaucoup de calculs. Elle peut demander beaucoup de traitements, donc elle peut durer très longtemps. Exemple: effectuer au 1er degré l'expression -((-3))*2/3*a^2*(-2*b^56)*b*(((-a^12))).

Pour éviter un test lourd pour voir si une expression est effectuée ou non, un champs booléen "effectué" est ajouté à chaque noeud. Il vaudra vrai si l'expression dont le noeud en question est la racine, est effectuée au 1er degré, faux sinon.

3.6.Le champs "référence canonique"

Pour établir l'équivalence mathématique de deux expressions, il est nécessaire de comparer leurs expressions canoniques équivalentes. Nous considérons que deux expressions sont équivalentes si leurs expressions canoniques sont égales (point de vue chaîne de caractères) à la commutativité près: b*a+c est égal à c+a*b, à la commutativité près.

Toute canonisation peut prendre de grande proportion point de vue temps. Il est, donc, inutile de recommencer ce long calcul. C'est pourquoi un champs "référence canonique" est ajouté à chaque noeud. Soit N un noeud quelconque, Exp l'expression dont n est la racine, ce nouveau champs indique si l'expression canonique Can associée à Exp a été calculée auquel cas il référence la chaîne de cararactères qui représente Can.

4.REPRESENTATION DES SOLUTIONS

4.1.Analyse du problème

D'un point de vue externe, une solution est une suite ordonnée d'expressions équivalentes dont la première est l'énoncé et la dernière une expression factorisée. Une solution n'est pas unique: en toute généralité, plusieurs combinaisons de méthodes sont possibles sur le même énoncé.

La représentation de la solution devra donc exprimer l'enchaînement des différentes transformatios mais aussi donner les moyens nécessaires pour connaître quels sont les chemins déjà découverts et quels sont ceux qu'il reste à essayer.

Elle devra contenir les renseignements sur la méthode de factorisation qui a permis de passer d'une expression à une autre: donc identifier la méthode mais aussi le numéro de l'étape dans la méthode (v. artifice, groupement).

4.2.Le champs "méthode" et le champs "étape"

Soit Exp une expression ou une sous-expression quelconque, N son noeud racine. Le noeud N comprend un champs "méthode" qui contient un identificateur de la méthode appliquée à Exp. Si on a appliqué aucune transformation sur Exp, le champs "méthode" doit l'indiquer. Grâce à cette structure, les exigences (renseignements sur la méthode appliquée) du paragraphe précédent sont remplies.

4.3.Le champs "référence solution"

On peut imaginer de chaîner les noeuds racines de chaque solution mais il ya mieux à faire. Lorsque l'expression initiale est un produit, une puissance ou un moins, on procède à un "éclatement" en sous-expressions sur lesquels on essaie réllement d'appliquer les méthodes de factorisation. Exemple d'éclatement: si on veut factoriser le produit a*b*c, on essaie de factoriser a, b, c et ensuite on fait le produit des résultats (v. chapitre III, § 2). On va utiliser la structure de produit, puissance ou moins existant déjà dans l'expression initiale. On va calquer la représentation des solutions sur la structure de l'énoncé.

Soit Exp une expression, N le noeud racine de Exp, M la méthode appliquée sur Exp, E l'étape dans la méthode. Le noeud N contient un champs "référence solution" qui référence le noeud racine de l'expression résultat de l'application de la méthode M, étape E, sur Exp. Si Exp n'a subit aucune méthode de factorisation, alors le

champs "référence solution" doit l'indiquer.

4.4. Description de la représentation des solutions

Soient Exp une expression dont N est le noeud racine,

si Exp est un produit (puissance, moins)
"Eclater", c'est à dire passer aux enfants F1, F2, ... Fn. Si une solution à la factorisation de Exp existe, alors elle est le produit des factorisations des enfants F1, F2, ...Fn.

```
N: type: produit
   reffr:pas
   reffi:F1
   méthode: pas
   étape:   pas
   ref-sol: pas
```

Si Exp est une somme alors
si ref-sol: pas (méthode: pas, étape: pas) alors
il n'y a pas eu d'application de méthodes sur Exp.
si ref-sol: N1 (N1 est le noeud racine de Exp1)
alors

```
N : type: somme
    reffi:?
    reffr:pas
    refsol: N1
    méthode:M
    étape:  E
```

Exp1 est le résultat de l'application de la méthode M, étape E sur Exp.

Analyse de Exp1:
si ref-sol: pas alors
pas d'alternative et pas de poursuite de méthodes à étapes
si ref-sol: N2 (N2 est le noeud racine de Exp2)
alors

```
N1: type: ?
    reffi:?
    reffr:?
    méthode: M1
    étape:   E1
    ref-sol: N2
```

Exp2 est le résultat de l'application de la méthode M1, étape E1, sur Exp.

Exemple:
énoncé: 4*a^2-4*b^2
1ere solution: m-é-é: 4*(a^2-b^2)
f-rem: 4*(a-b)*(a+b)

2de solution: f-rem: (2*a-2*b)*(2*a+2*b)

m-é-é: 2*(a-b)*2*(a+b)

```
père:type: somme
     chcar:4*a^2-4*b^2
     reffi: ...
     reffr: pas
     ref-sol: s1
     méthode: m-é-é
     étape  : pas

1ere solution:

s1   :type: produit
     chcar: 4*(a^2-b^2)
     reffi: f1
     reffr: pas
     ref-sol: s2
     méthode: f-rem
     étape  : pas

"éclatement"

  f1:  type: valeur
       chcar: 4
       reffi: pas
       reffr: f2
       ref-sol: pas
       méthode: pas
       étape:   pas

  f2:  type:  somme
       chcar: (a^2-b^2)
       reffi: ...
       reffr: s1
       ref-sol: s11
       méthode: f-rem
       étape:   pas

s11: type: produit
     chcar:(a-2)*(b-2)
     reffi: ...
     reffr: pas
     ref-sol: pas
     méthode: pas
     étape:   pas

1ere solution:

s2:  type:  produit
     chcar: (2*a-2*b)*(2*a+2*b)
     reffi: f21
     reffr: pas
     ref-sol: pas
     méthode: pas
     étape:  pas
```

"éclatement"

```
f21: type: somme
     chcar: (2*a-2*b)
     reffi: ...
     reffr: f22
     réf-sol: s21
     methode: m-é-é
     étape:   pas

f22: type: somme
     chcar: (2*a+2*b)
     reffi: ...
     reffr: s2
     ref-sol: s22
     méthode: m-é-é
     étape:   pas

s21: type: produit
     chcar: 2*(a-b)
     reffi: ...
     reffr: pas
     ref-sol: pas
     méthode: pas
     étape:   pas

s22: type: produit
     chcar: 2*(a+b)
     reffi: ...
     reffr: pas
     ref-sol: pas
     méthode: pas
     étape  : pas
```

4.5.Critiques

Cette manière de voir la représentaion des solutions implique que l'on gagne du temps, de la place mémoire, de la simplicité lorsqu'on veut construire une autre solution mais implique plus de temps lorsqu'on veut construire la suite d'expressions représentant le cheminement de factorisation: en effet, il faut descendre dans l'arbre initial en "éclatant" pour les noeuds produits, puissances, moins, en "chaînant" (avec le champs ref-sol) pour les étapes successives et pour les solutions alternatives.

5.CONCLUSION

5.1.Implémentation

5.1.1.L'arbre fils-frère

Soient un tableau de nn noeuds,
chaque noeud = un record,
le champs type du noeud: énumère
(pl,mu,pu,val,vari,mo)
les références au fils et au frère:
intervalle -nn..nn
pas de référence: 0
référence au père: -père

5.1.2.Implémentation du champs "chaîne de caractères"

soient deux champs du record noeud:
long-ch: longueur de la chaîne: 1..nc
ref-ch: référence dans un tableau de
caractères: intervalle 1..nc
où nc est le nombre de caractères du
tableau

5.1.3.Implémentation du champs "ordre initial"

Soit un champs du record noeud: ordinit:
integer
Remarque: l'ordre total sur les types de
noeuds est défini par l'énumération (v. 5.1.1)

5.1.4.Implémentation du champs "effectué 1er degré"

Soit un champs du record: effectué:boolean

5.1.5.Implémentation du champs "référence canonique"

Soit un champs variable du record: refcan:
boolean.
Soient deux sous-champs:
chcan: référence dans le tableau des
caractères: 1..nc
lchcan: longueur de chaîne: 1..nc .

5.1.6.Implémentation des solutions

Soient trois champs du record noeud:

refsol: référence solution: 0..nn
pas de référence: 0
méthode: numéro d'identification de
méthode: intervalle 0..nm
où nm est le nombre de méthodes
pas de méthode : 0
étape: numéro de l'étape dans la méthode:
integer.

5.2.Gestion mémoire

Les deux tableaux (de noeuds, et de caractères) déjà introduits sont, en réalite des "stacks". Les deux tableaux ont une seule dimension. Quatre variables sont nécessaires à la gestion des deux stacks:

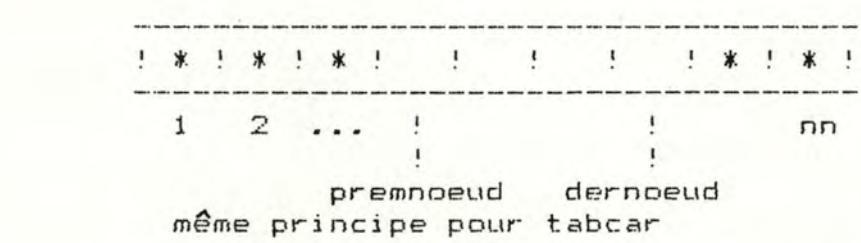
premcar: premier élément vide du tableau de caractères

dercar: dernier élément vide du tableau de caractèrers

premnoeud: premier élément vide du tableau de noeuds

dernoeud: dernier noeud vide du tableau de noeuds.

Illustration: tabnoeud



même principe pour tabcar

Plutôt que d'utiliser les pointeurs du Pascal entre les records noeuds, nous avons décidé de gérer nous-même la place mémoire un utilisant des stacks sous forme de tableaux.

5.3.Déclaration Pascal

```
const nn = 200;
      nc = 400;
      nm = 7;


type typenoeud = (pl,mu,pu,val,vari,mo);
     refnoeud = -nn..nn
     refcar = 1..nc
     noeud = record
               tn: typenoeud;
```

```
                reffi: refnoeud;
                reffr: refnoeud;
                longch:refcar;
                refch: refcar;
                ordrinit:integer;
                effectue: boolean;
                refsol:refnoeud;
                méthode: 0..nm;
                étape: integer;

                case refcan: boolean of
                  false:();
                  true: (chcan: refcar;
                         lchcan:refcar)
                end;

var tabnoeud: packed array 1..nn of noeud;
    tabcar: packed array 1..nc of char;
    premnoeud, dernoeud, premcar, dercar: integer;
```

Chapitre V

IMPLEMENTATION DE LA MISE EN EVIDENCE

1. Introduction
2. Fonctions implémentées
3. Le comment
4. Algorithme de M-E-E
5. Description de l'architecture physique

---

1. INTRODUCTION

Sur la base des structures de données constuites au chapitre III, il est possible, maintenant, d'entreprendre l'implémentation du module de factorisation qui fera partie du didacticiel.

En utilisant les enseignements de notre analyse de la factorisation, nous avons implémenté la mise en évidence. Le code ainsi que les spécifications concrètes des fonctions implémentées se trouvent en Annexe 4.

2. FONCTIONS IMPLEMENTEES

Pour implémenter le module-méthode M-E-E, il nous faut aussi implémenter une saisie d'expressions à l'écran, la construction de la réprésentation interne, une suites d'utilitaires pour comparer et construire des expressions.

2.1. Saisie d'une expression à l'écran

Objet: saisir à l'écran une suite de caractères; corriger syntaxiquement la chaîne saisie; diagnostiquer les éventuelles erreurs; reprendre la saisie si il y a erreur. (v. Annexe 4.2.)

2.2. Constuction de l'arbre fils-frère

Objet: construire la structure d'arbre fils-frère à partir d'une chaîne de caractères correcte syntaxiquement. (v. Annexe 4.3.)

2.3. Ordonner l'arbre fils-frère

Objet: ordonner l'arbre fils-frère, transformer celui-ci en représentation interne d'une expression. (v. Annexe 4.4.)

2.4. Utilitaires

Objet: établir l'égalité de chaînes de caractères, transformer un nombre entier en une chaîne de caractères, transformer une chaîne en nombre, construire une expression identique, construire des expressions produits et puissance. (v. Annexe 4.5. et 4.6.).

3.LE COMMENT

Pour la gestion de place mémoire, on sait déjà que nous utilisons deux stacks (de caractères et de noeuds); chaque bloc qui voudra utiliser de la place mémoire sera une fonction booléenne qui renvoie vrai si il y a suffisamment de place et faux sinon. Ainsi, les spécifications des sous-modules et des primitives de base données au chapitre III sont légèrement modifiées. Leur fondement ne change pas; il faut seulement tenir compte de la taille des stacks et arreter le bloc en cours s'il y a manque de place pour mener à bien l'objet du bloc.

Scénario du sous-système implémenté:

Dans notre sous-système, l'utilisateur tape l'expression à factoriser à l'écran. Celle-ci est lue, corrigée syntaxiquement. Ensuite, le programme construit sa réprésentation interne: arbre fils-frère ordonné. En dernier lieu, les facteurs communs s'il y en a sont mis en évidence. Le résultat apparaît sous forme d'une expression si la mise en évidence a réussi, sous forme d'un message tel que "manque de place" ou "échec m-é-é" sinon.

4.ALGORITHME DE M-E-E

Ci-dessous l'algorithme de mise en évidence conçu avant de construire les structures de données et la gestion de la mémoire. Il ne tient aucun compte d'un éventuel manque de place. Ainsi, la stratégie de mise en évidence apparaît plus clairement.

```
begin
  expf := un; expr := expd;
  expi := expd; m-e-e := true;
  while get-terme(expi,terme) do
   while get-fact(terme,f) do
     begin
       expb := base(f);
       d := degre-q (expr,expb);
       if d <> 0 then
         begin
           fc := expos(expb,d);
           expr := quot (expr,fc);
           expf := prod (expf,expd)
         end
     end;
  if expf = un then m-e-e := false
  else expf := prod (rxpf,expr)
end;
```

Pour les spécifications de M-E-E et des primitives v. chapitre III § 4. Pour degré-q:

```
Function DEGRE-Q (expd,expf: expression): integer;
```

Prcd: expd,expf sont des expressions.

Pscd:degré-q vaut la multiplicité de expf em tant que facteur commun aux termes de expd.

5.DESCRIPTION DE L'ARCHITECTURE PHYSIQUE

Le code se découpe en 5 "UNITS" (unité de compilation dont le code peut être non résident en mémoire à l'exécution).

Le unit GLOBAL contient toutes les variables globales de gestion de mémoire et celles nécessaires à la représentation interne des expressions.

Le unit SUNBROTHER contient l'implémentation de la construction de l'arbre fils-frère.

Le unit ORDONNER contient l'implémentation de l'ordonnancement de l'arbre fils-frère.

Le unit UTILITAIRES contient l'implémentation de l'égalité de chaînes de caractères, de la construction des expression UN et ZERO, de la transformation d'un entier en une chaîne de caractères et vice versa, de duplication d'expression (construction d'une expression de structure identique).

Le unit MISEENEVIDENCE contient l'implémentation de degre-q, quot, construction de produits et de puissances, et enfin, la mise en évidence.

Pour chaque unit, il existe un programme de test, à savoir: testlire, testfilsfrere, testutilitaires, testmee (test ordre se trouve dans testutilitaires).

CONCLUSION

Déjà lorsque nous dressions les grandes lignes du mémoire, un tel projet m'enthousiasmait, bien que je savais qu'il présentait une part de risque. La première solution (langage d'auteur) était possible. Celle que nous avions choisie était peut-être possible.

Nous nous sommes rendu compte que notre approche était possible après une lente et profonde analyse de ces fameux mécanismes de factorisation. Vers le milieu de l'année, nous nous sommes fixé l'objectif d'implémenter la mise en évidence et d'améliorer les performances de celle-ci.

L'analyse fut particulièrement longue parce que je savais que les "backtrackings" sur les décisions de conception peuvent devenir catastrophiques. La mise en évidence n'a "tourné" que début août. Il me restait à rédiger et à éditer le mémoire. Je n'ai donc pas eu le temps de rapporter et d'améliorer les performances du sous-système implémenté.

Je regrette, aussi de ne pas pouvoir ajouter le module directeur pour la factorisation. Je l'ai imaginé; malheureusement, il restera à l'état de projet ou sera implémenter par quelqu'un d'autre.

Je tiens à terminer par mes satisfactions. J'ai été enthousiasmé par l'approche que nous avons eue du problème, par son côté "risque" et surtout par la démarche intellectuelle que supposait cette modélisation.

ANNEXE

ANNEXE 1 : DEFINITION DE LA FACTORISATION : EXERCICES TYPIQUES

Annexe 1

DECOMPOSITION EN FACTEURS : SYNTHESE

1° Mettre les facteurs communs en évidence

2° Songer :

- aux identités remarquables

$$a^2 - b^2 = (a+b)(a-b)$$
$$a^2 \pm 2ab + b^2 = (a \pm b)^2$$
$$a^2 + b^2 + c^2 + 2ab + 2bc + 2ac = (a+b+c)^2$$
$$a^3 \pm b^3 = (a \pm b)(a^2 \mp ab + b)^2$$
$$a^3 \pm 3a^2b + 3ab^2 \pm b^3 = (a \pm b)^3$$

- au trinôme du second degré

1° méthode : $x^2 + bx + c = (x-p)(x-q)$
$= x^2 - (p+q)x + pq$
$p + q = -b \quad p.q = c$
si $ax^2 + bx + c$
alors mettre a en évidence et procéder de même.

2° méthode : $ax^2 + bx + c = o$
$\Delta = b^2 - 4ac \longrightarrow x', x''$
$ax^2 + ba + c = a(x-x')(x-x'')$

3° méthode : artifice pour décomposer $x^2 + bx + c$, ajouter et retrancher le terme qui forme avec $x^2 + bx$ un carré parfait.

3° Songer à grouper des termes pour faire apparaître un facteur commun pour employer les identités remarquables.

4° Songer à : ajouter et retrancher un terme,
dédoubler un terme puis grouper,
effectuer puis grouper.

Annexe 1

5° Songer à : méthode des diviseurs du type $x-a$ ou $ax+b$
(division de Horner)

6° Songer aux quotients remarquables

m entier positif : . $x^m - a^m$ est divisible par $x-a$

. $x^m + a^m$ est divisible par $x+a$
ssi m impair

. $x^m - a^m$ est divisible par $x+a$
ssi m pair

Annexe 1

EXEMPLES REPRESENTATIFS DES DIFFERENTES METHODES(CFR SYNTHESE)

1° Mise en évidence

- $x^2 + x = x(x+1)$

2° Identités remarquables

- $x^4 - y^4 = (x^2-y^2)\ (x^2+y^2) = (x-y)\ (x+y)\ (x^2+y^2)$
- $x^6 - 2x^3 + 1 = (x^3-1)^2 = (x-1)^2\ (x^2+x+1)^2$
- $x^9 - 8 = (x^3-2)\ (x^6+2x^3+4)$
- $x^2 + 4 + y^4 + 4x + 4y^2 + 2xy^2 = (x+2+y^2)^2$
- $y^3 + 3y^2 + 3y + 1 = (y+1)^3$

3° Trinôme du second degré

1°méthode

- $x^2 - 3x + 2 = (x-1)\ (x+1)$

  $p+q = 3$

  $p.q = 2$

- $2x^2 - 2x - 4 = 2(x^2-x-2) = 2(x+1)\ (x-2)$

  $p+q = 1$

  $p.q = 2$

2°méthode

- $-x^2 + 2x + 3 = 0$ $\qquad -x^2 + 2x + 3 = -(x-3)(x+1)$

  $\Delta = 16$ $\quad x' \\ x''$ $\Big\}\ -2 \pm 4/-2 \begin{matrix} \nearrow 3 \\ \searrow -1 \end{matrix}$

3°méthode

- $x^2 + x + 1/6 = x^2 + x + 1/4 - 1/4 + 1/6 = (x+1/2)^2(1/12\hat{}1/2)^2$

  $= (x+1/2-1/12\hat{}1/2)(x+1/2+1/12\hat{}1/2)$

4° Groupements

- $x^2 - 4x^6 - 2x^5 + 8x^3 + x^2 - 4 = x^6(x^2-4)+(x2-4)-2x^3(x^2-4)$
  $= (x^2-4)(x^6+1-2x^3)$
  $= (x^2-4)(x^3-1)^2$
  $= (x-2)(x+2)(x-1)^2$
  (x^2+x+1)

- $a^2 - 2ab + b^2 - 1 = (a-b)^2-1$
  $= (a-b-1)(a-b+1)$

5° Artifices

- $x^4 + y^4 = x^4 + y^4 + 2x^2y^2 - 2x^2y^2$
  $= (x^2+y^2)^2 - (2\hat{}1/2\ xy)^2$
  $= (x^2+y^2-2\hat{}1/2\ xy)\ (\ x^2+y^2+2\hat{}1/2\ xy)$
- $x^3 + 4x + 5 = x^3 + 4x + 4 + 1$
  $= (x+1)(x^2-x+1)+ 4(x+1)$
  $= (x+1)(x^2-x+1+4) = (x+1)(x^2-x+5)$
- $(a+1)(a-2) + 2a^3 = a^2 + a - 2a - 2 + 2a^3$
  $= a^2 - a + 2a^3 - 2$
  $= a(a-1) + 2(a^3-1)$
  $= a(a-1) + 2(a-1)(a^2+a+1)$
  $= (a-1)(a+2a^2+2a=2) = (a-1)(2a^2+3a+2)$

6° Diviseurs binomes (x-a)

- $x^4 - 2x^2 + 3x - 2 = P(x)$ coefficients entiers

racines entières : diviseurs positifs ou négatifs de -2

$\{+2,-2,-1,+1\}$

$P(1) = o = P(-2)$

"Horner"

$x^4 - 2x^2 + 3x - 2$
$= (x-1)(x+2)(x^2-x+1)$

	1	0	-2	3	-2
1		1	1	-1	2
	1	1	-1	2	0
-2		-2	+2	-2	
	1	-1	1	0	

Si pas de racines entières, alors aucune méthode de recherche n'est disponible au-dessus du 2°degré

7° Quotients remarquables

- $x^4 - 16 = (x-2)\ (x^3+2x^2+4x+8)$

	1	0	0	0	-16
2		2	4	8	16
	1	2	4	8	0

- $x^7 + y^7 = (x+y)\ (x^6-yx^5+y^2x^4-y^3x^3+y^4x^2-y^5x+y^6)$

	1	0	0	0	0	0	0	y^7
$-y$		$-y$	y^2	$-y^3$	y^4	$-y^5$	y^6	$-y^7$
	1	$-y$	y^2	$-y^3$	y^4	$-y^5$	y^6	0

Annexe 1

SERIES D'EXERCICES

1° Mise en évidence

1. $(a+1)^2 + 2(a+1) = (a+1)(a+1+2) = (a+1)(a+3)$
2. $6a^2b - 2a^2b^3 + 8ab^2 = 2ab(3a-ab^2+4b)$
3. $3(2-x)^2 - 3(x-2)^3 = (2-x)(3(2-x)+3(x-2)$
   $= (2-x)(6-3x+3x^2-12x+12)$
   $= (2-x)(3x^2-15x+18)$
4. $2(x-1)(a+b) + a(1-x) = (x-1)(2a+2b-a)$
   $= (x-1)(a+2b)$
5. $2a^2 - 3a - 2a^3 + 3a^2 = a(2a-3-2a^2+3a)$
   $= a(2a-3-a(2a-3))$
   $= a(2a-3)(1-a)$
6. $(x-1)(x^2-4) - (x-1)(x-2) +5(x-1) = (x-1)(x^2-4-x+2+5)$
   $= (x-1)(x^2-x+3)$

2° Identités remarquables

- $a^2 - 9b^2 = (a-3b)(a+3b)$
  $(a-b)^2 - a^2 = (a-b-a)(a-b+a) = -b(2a-b)$
  $a^6b^4 - a^2 = (a^3b^2-a)(a^3b^2+a)$
  $3/5x^2 - 5/3y^2 = (3/5\hat{}1/2x-5/3\hat{}1/2y)(3/5\hat{}1/2x+5/3\hat{}1/2y)$
  $4(a+b)^2 - 9b^2 = (2a+2b-3b)(2a+2b+3b) = (2a-b)(2a+5b)$
- $1 - 6x + 9x^2 = (1-3x)^2$
  $2a^2b - b^2 - a^4 = -(a^2-b)^2$ $(a-b)^2 + 2x(a-b) + x2$
  $= (a-b+x)^2$
  $a^4 + a^2 + 1/4 = (a^2+1/2)^2$
- $8x^3 - 1 = (2x-1)(4x^2+2x+1)$
  $x^3 + 1/27 = (x+1/3)(x^2-x/3+1/9)$
  $(x-y)^3 - y^3 = (x-y-y)((x-y)^2+(x-y)y+y^2$
  $= (x-2y)(x^2-2xy+y^2+xy+y^2+y^2)$
  $= (x-2y)(x^2-xy+y^2)$
  $1-8(a+b)^3 = (1-2(a+b))\ (1+2(a+b)+4(a+b)^2)$
  $= (1-2a-2b)\ (1+2a+2b+4a^2+8ab+4b^2)$
- $8 - 12a + 6a^2 - a^3 = (2-a)^3$
  $a^3 + a^2b + ab^2/3 + b^3/27 = (a+b/3)^3$

3° Trinome du second degré

- $x^2 - 5x + 6 = (x-2)(x-3)$

  $p+q = 5$

  $p.q = 6$

  $x^2 - 3x - 18 = (x+3)(x-6)$

  $x^2 - x - 42 = (x+6)(x-7)$

  $x^2 - 28 - 3x = (x+4)(x-7)$

  $x^2 - 7x + 12 = (x-4)\ (x-3)$

  $2x^2 - 12x - 14 = 2(x+1)(x-7)$

  $7x^2 - 49x - 126 = 7(x+2)(x-9)$

  $11x^2 - 2(19x+12) = 11(x-4)(x+6/11)$

  $3/4x^2 + 2x + 4/3 = 3/4(x+4/3)^2$

  $7x^2 - 45 + 58x = 7(x+9)(x-5/7)$

  $6x^4 + 5x^2 + 1 = 6x^2 + 5x + 1$

  $- = 25-24=1 \quad = 6(x+1/2)(x+1/3)$

  $-5\pm1/12 \quad -1/2$

  $-1/3 \qquad = 6(x^2+1/2)(x^2+5/3)$

  $2x^2 + 3x - 2 = 2(x-1/2)(x+2)$

  $18x^2 + 3x - 1 = 18(x-16)(x+1/3)$

  $63x^2 + 25x + 2 = 63(x+36/126)(x+14/126)$

  $3x^2 - 5x + 1 = 3(x-5-13\hat{}1/2/6)(x-5+13\hat{}1/2/6)$

- $x^2 - 10x + 8 = x^2 - 10x + 25 - 25 + 8$

  $= (x-5)^2 - 17$

  $= (x-5-17\hat{}1/2)(x-5+17\hat{}1/2)$

  $2x^2 - 14x + 2 = 2(x^2-7x+1)$

  $= 2(x^2-2(7/2)x+ 49/4-49/4+1)$

  $= 2((x-7/8)^2-45/4) = 2(x-7-45\hat{}1/2)$

  $(x-7+45\hat{}1/2)$

  $3x^2 - 18x + 5 = 3(x^2-6x+5/3)$

  $= 3(x^2-6x+9-9+5/3)$

  $= 3((x-3)^2-22/3)$

  $= 3(x-3-22/3)(x-3+22/3) = 3(x-31/3)(x+13/3)$

$$x^2 - 5x + 2 = x^2 - 2(5/2)x + 25/4 - 25/4 + 2$$
$$= (x-5/2)^2 - 17/4$$
$$= (x-5/2-17\hat{}1/2/2)(x-5/2+17\hat{}1/2/2)$$

4° Groupements de termes

- $a^2 - 2ab + b^2 - 1 = (a-b)^2 - 1 = (a-b+1)(a-b-1)$

$4x^2 + 2x - 9y^2 - 3y = 4x^2 - 9y^2 + 2x - 3y$
$= (2x-3y)(2x+3y) + 1(2x-3y)$
$=(2x-3y)(2x+3y+1)$

$x^3 + 1 + xy + y = (x+1)(x^2-x+1) + y(x+1)$
$= (x+1)(x^2-x+1+y)$

$a^3 - a^2 - b^3 + b^2 = (a^3-b^3) - (a^2-b^2)$
$= (a-b)(a^2+ab+b^2) - (a-b)(a+b)$
$= (a-b)(a^2+ab+b^2-a-b)$

$x^4 - 2x^3 + x - 2 = x^3(x-2) + 1(x-2) = (x-2)(x^3+1)$
$= (x-2)(x+1)(x^2-x+1)$

$x^4 - 1 + 2x^3 + 2x^2 = (x^2-1)(x^2+1) + 2x^2(x+1)$
$= (x+1)((x-1)(x^2+1)+2x^2)$
$= (x-1)(x^3-x^2+x-1+2x^2)$
$= (x-1)(x^3+x^2+x-1)$

$b^2y + a^2y - b^2 - a^2 = (b^2+a^2)y - (b^2+a^2)$
$= (b^2+a^2)(y-1)$

$y^3-b^2y - b^3 + by^2 + y^2 - b^2 = y(y^2-b^2) + (y^2-b^2)b + 1(y^2-b^2)$
$= (y^2-b^2)(y+b+1)$
$= (y-b)(y+b)(y+b+1)$

5° Groupements avec artifice

- $a^4 + b^4 = a^4 + b^4 + 2a^2b^2 - 2a^2b^2$
  $= (a^2+b^2)^2 - (2\hat{}1/2\ ab)^2$
  $= (a^2=b^2-2\hat{}1/2\ ab)(a^2+b^2+2\hat{}1/2\ ab)$

$x^3 + 2x - 3 = x^3 - x + 3x - 3$
$= x(x^2-1) + 3(x-1)$
$= (x-1)(x(x+1)+3)$
$= (x-1)(x^2+x+3)$

$x^4 + y^4 + x^2y^2 = x^4 + y^4 + 2x^2y^2 - x^2y^2$
$= (x^2+y^2-xy)(x^2y^2+xy)$

$(a-b)^2 + 4ab = a^2 + b^2 - 2ab + 4ab$
$= a^2 + b^2 + 2ab$
$=(a+b)^2$

$x^4y - 3x^2y + 2xy = x^4y - x^2y - 2x^2y + 2xy$
$= x^2y(x^2-1) - 2xy(x-1)$
$= (x-1)(x^2y(x+1)-2xy)$
$= (x-1)(x^3y+x^2y-2xy)$

$a^8 + a^4 - 2 = a^8 + 2a^4 - a^4 - 2$
$= a^4(a^4-1) + 2(a^4-1)$
$= (a^4-1)(a^4+2)$
$= (a^2-1)(a^2+1)(a^4+2)$
$= (a-1)(a+1)(a^2+1)(a^4+2)$

$c(a^2-c) +a(a-c^2) = a^2c-c^2 + a^2 - ac^2$
$= ac(a-c)+ a^2-c^2$
$= (a-c)(ac+a+c)$

$4x^4 + y^4 + 3x^2y^2 = 4x^4+y^4 + 4x^2y^2 -x^2y^2$
$= (2x^2+y^2-xy)(2x^2+y^2-xy$

$(10x^4 - 41)^2 - (6x^4-40) = ((6x^4-40) + (4x^4-1))^2-(6x^4-40)^2$
$= (6x^4-40)^2 + (4x^4-1)^2$
$+ 2(6x^4-40)(4x^4-1)-(6x^4-40)^2$
$= (4x^4-1)(4x^4-1+12x^4-80)$
$= (4x^4-1)(16x^4-81)$
$=(2x^2-1)(2x^2+1)(4x^2-9)(4x^2+9)$
$= (2\hat{}1/2x-1)(2\hat{}1/2x+1)(2x^2+1)$
$(2x-3)(2x+3)(4x^2+9)$

$$24\,x^2y^2 - 1 - 2xy = 16\,x^2y^2 + 8x^2y^2 - 1 - 2xy$$
$$= (4xy-1)(4xy+1) + 2xy(4xy-1)$$
$$= (4xy-1)(4xy+1+2xy)$$
$$= (4xy-1)(6xy+1)$$
$$x(x^2+y) - y(y^2+x) = x^3 + xy - y^3 - xy$$
$$= x^3 - y^3 = (x-y)(x^2+xy+y^2)$$

6° Méthode des diviseurs "x-a"

1. $x^3 - x^2 - 4x + 4 = (x-2)(x^2+x-2) = (x-2)(x-1)(x+2)$

$\{\pm1, \pm2, \pm4\}$

2. $x^4 - 5x^2 + 8x - 12 = (x-2)(x+3)(x^2-x+2)$

$\{\pm1, \pm2, \pm3, \pm4, \pm6, \pm12\}$

3. $2x^4 - 12x^3 + 19x^2 - 13x + 4 = (x-1)(x-4)(2x^2-2x+!)$

$\{\pm1, \pm2, \pm4\}$

4. $6x^4 - 5x^3 - 23x^2 + 20x - 4 = (x-2)(x+2)(6x^2-5x+1)$

$- = 1$

$6(x-1/2)(x-1/3)$

$= 6(x-2)(x+2)(x-1/2)(x-1/3)$

5. $x^3 + 9x^2 + 11x - 21 = (x+3)(x^{-}+7)(x-1)$

$\{\pm1, \pm3, \pm7, \pm21\}$

6. $2x^3 - 7x^2 + 2x + 3 = (x-3)(2x^2-5x-3)$
$= (x-3)2(x-3)(x-1/2)$

7° Quotients remarquables

$$a^5 - b^5 = (a-b)(a^4+ba^3+b^2a^2+b^3a+b^4)$$

						$-b^5$
	1	0	0	0	0	
b		b	b^2	b^3	b^4	b^5
	1	b	b^2	b^3	b^4	0

$$ax^8 - a = a(x^8-1)$$
$$= a(x-1)(x^7+x^6+x^5+x^4+x^3+x^2+x+1)$$

Annexe 1

$$x^5 + 32 = (x+2)(x^4-2x^3+4x^2-8x+16)$$

	1	0	0	0	0	32
-2		-2	4	-8	16	-32
	1	-2	4	-8	16	0

$$a^6-b^6 = (a-b)(a^5+ba^4+b^2a^3+b^3a^2+b^4a+b^5)$$

$$32x^5+243 = (x+3)(32x^4\ldots\ldots\ldots\ldots\ldots)$$

Annexe 1

Devant un polynôme à factoriser:

MODULE A : polynôme $a_n x^n + a_{n-1} x^{n-1} + \ldots + a_o$.

Mettre x, x^2,.... en évidence (si pas de terme indépendant)

si degré = 2 ,alors méthodes disponibles :

- somme, produit (si coefficients entiers)
- ajout et retrait (si carré parfait proche)
- Δ dans les autres cas

sinon (polynôme de degré 3 et plus).

1°recherche: suivant nombre de coefficients

- si le polynôme ne contient que terme indépendant et terme en x^n, alors quotient remarquable
- si le polynôme ne contient que T.Ind. et T. en x^{2n} + T. intermédiaire, alors:
  - (1)- ? en posant $x^n = X$ (second degré)
  - (2)- ? décomposer le terme indépendant ou le terme intermédiaire pour faire apparaître un facteur commun.
  - (3)- ? ajouter ou retirer un terme pour faire apparaître un Produit remarquable.
- si plusieurs termes intermédiaires, alors même recherche (2) et (3) mais plus le nombre de termes Int. augmente, plus la recherche se complique.

2°recherche : correspondance de coefficients

En se basant sur le nombre de termes du polynôme(pair ou impair) :

- dénicher une suite de coefficients(de rapport constant)
  ex: $0x^2 - 4x^6 - 2x^5 + 8x^3 + 0x^2 - 4$ (4, 4, 4)
- dénicher cette correspondance à l'aide d'une décomposition ou d'un artifice: ex : $x^4 + 5x^2 + 4$ ($5 = 4+1$)

Annexe 1

3°recherche diviseurs "x-a"

Méthode systématique

MODULE B: polynôme mis sous forme de parenthèse

- Mettre en évidence
  ex: $(x-2)(x-3) + (x-2) = (x-2)(x-3+1)$
- Passer en revue les produits remarquables et quotients remarquables
  ex: $(x+2)^2 - 9$
- Groupement et factorisation permettant d'aborder la formule de Produits Remarquables
  ex: $(x-1)^5 - x^5 + 1$
  se baser sur recherche (2) et (3) module A.

Au dernier moment : effectuer et donc revenir au module A.

ANNEXE 2 : SCENARIO

Annexe 2

En début de programme :

une explication apparaît à l'écran:

" Ce programme a pour but de t'apprendre à factoriser des polynômes en x ; il est divisé en 3 étapes de difficultés croissantes " .

NIVEAU 1 : " tu apprends à utiliser les différentes méthodes."

NIVEAU 2 : " tu apprends à factoriser des polynômes suivant l'une ou plusieurs méthodes vues au niveau 1."

NIVEAU 3 : " tu rivalises avec l'ordinateur: qui est le plus fort en factorisation, toi ou l'ordinateur ?"

" Ne vise pas trop haut !
Commence donc par le niveau 1 ou 2 ; ensuite je te conseillerai lorsque le moment sera venu pour toi, de passer au niveau supérieur " .

" Voilà : choisis ton niveau : 1,2 ou 3 ".

Niveau 1 : " tu apprends les différentes méthodes mises à ta disposition :

en voici la liste avec un exemple résolu :

1. mise en évidence ex: .....
2. identités remarquables ex: ....
3. second degré ex: ....
4. groupements ex: ....
5. artifices ex: ....
6. diviseurs x-a ex: ....
7. quotients remarquables ex: ....
8. menus de départ

Quelle méthode veux-tu travailler?"

Pour chaque méthode:

- un exemple de résolution avec explications sommaires
- répéter : - exercice de difficulté croissante
  - si factorisation non correcte, alors l'ordinateur signale l'erreur
  - reprendre l'ex. à l'endroit incorrect jusqu'à ce que 4 exercices consécutifs soient corrects
- l'ordinateur envoie le massage suivant :
  " tu peux passer à une autre méthode de résolution " .

Revenir à la liste.

Niveau 2 : " tu apprends à factoriser des polynômes suivant les méthodes vues en 1 ":

un message apparaît :

" Je vais te proposer des polynômes que tu devras factoriser suivant des méthodes vues au niveau 1"

" Je peux te proposer des exercices de 4 niveaux de difficultés :

facile	1
moyen	2
difficile	3
très difficile	4

1. : application d'une seule méthode : simple
2. : application d'une seule méthode : difficile
3. : application de deux méthodes : simple
4. : application de deux méthodes ou plus : difficile

" Ne vises donc pas trop haut : il n'y a aucune honte à commencer par le niveau facile, je te dirai à quel moment tu pourras passer à une difficulté supérieure, mais je pourrai aussi te faire revenir en arrière si je te trouve trop peu préparé ".

Pour chaque niveau :

- répéter
- proposer un exercice
- si l'élève ne parvient pas à le factoriser, alors l'ordinateur donne des renseignements un à un
- si l'élève commet une erreur, alors l'ordinateur signale et l'élève corrige jusqu'à ce que: trop d'erreursou bon comportement. Si trop d'erreurs , alors retourner à un niveau de difficulté inférieure ; sinon passer à un niveau de difficulté supérieure.

Niveau 3 : " Tu vas essayer de rivaliser avec moi "

Un message explique le jeu :

" Tour à tour, l'élève et l'ordinateur propose un polynôme à factoriser :(au moins degré 3 et le hasard décidera qui commencera)".

- Si l'élève ne le factorise pas en moins de deux minutes ou de manière incorrecte, l'ordinateur propose sa solution et gagne, sinon l'élève gagne.

→ Si l'ordinateur ne parvient pas à le factoriser: il lance le message suivant:

" en modifiant un seul coefficient,nous devons factoriser le polynôme".

- Si l'élève ne le factorise pas en moins de deux minutes ou de manière incorrecte,l'ordinateur propose sa solution et gagne, sinon l'élève gagne.

→ Si l'ordinateur ne parvient pas à le factoriser, en modifiant un deuxième coefficient, nous devons factoriser le polynôme.
Ainsi de suite jusqu'à la factorisation.

- L'élève et la machine pourraient proposer chacun 3 polynôme.
- Le total maximum serait de 6 points.
- La machine rappellerait après chaque étape (chaque polynôme) le classement.
- Possibilités de rejouer (et donc de revenir aux niveaux inférieurs si la machine est trop supérieure).

Remarque: tout au long du programme : le dialogue entre l'élève et la machine est basé sur l'animation.

ANNEXE 3 : ERREURS HABITUELLES

Annexe 3

TYPES D'ERREURS

1) Inhérentes à chaque méthode
2) Générale
3) De calcul numérique et algébrique

1. a) Identités remarquables

- Ne pas connaître les formules
- Confondre des produits remarquables :
  ex: $x^2 - 4$ et $(x-2)^2$
- Erreur de signe: ex: $(-x-1)^2 = x^2 - 2x + 1$

b) Mise en évidence

- Confusion entre $1 - x$ et $x - 1$
- Erreur classique ex : $(x-3)(x^2-4)+(x-3)(x+2)+(x-3)$
  $= (x-3)(x^2-4+x+2)$

c) Trinôme du second degré

- Méthode du réalisant :
  - ne pas connaître les formules
  - ne pas ordonner les coefficien et donc de mal les employer
  - oublier le coefficient de x^2
    ex: $6x^2+ .. = ?(x..)(x..)$
  - ne pas calculer les deux racines correctement.
- Méthode somme et produit :
  - oublier de mettre a en évidence
  - $x^2-(p+q)x+pq = x^2-2x+1$ $\quad p+q = -2$
- Méthode ajout et retrait pour faire un carré parfait
  - très peu utilisée par les élèves

d) Groupements : /
e) Artifices : /

} cfr erreurs algébriques

f) Méthode diviseurs: x-a"

- oublier des coefficients dans la grille d'Horner
  ex: $2x^3-3x^2+x-1$ | 2 | -3 | 1 |
- oublier d'indiquer 0 si puissance de x absente
  ex: $2x^3-x+1$ | 2 | -1 | 1 |
- appliquer dans le cas de coefficients non-entiers la méthode de recherche des racines parmi les diviseurs du terme intermédiaire
- P(-1) = 0 +1

g) Quotients remarquables :

- factoriser $32x^5 + 243 = (2x)^5 + 3^5$:
  le factoriser comme $x^5 + 3^5$
- mal calculer le quotient à l'aide d'Horner
  (cfr. méthode diviseurs "x-a")

2. Générales :
   - ne pas comprendre le mot factoriser
     ex: factoriser $x^2 - 3x + 2$ et écrire $S= \{1,2\}$
   - simple erreur de transcription
     ex: 56 devient 96
   - confondre des coefficients lorsque le polynôme est ordonné

3. Numériques et algébriques :

- $-x^2+ 1 = x^2 - 1$
- $-(x+3) = -x + 3$
- calculer : $P(x) = -x^2$ x=1 P(-1)=1
- erreur de calcul de la valeur d'un polynôme pour un x donné.

```
(*=======================================

                    Annexe 4.

      SPECIFICATION ET CODE DE LA MISE EN EVIDENCE
      --------------------------------------------


=======================================

                Annexe 4.1.

                UNIT GLOBAL


=======================================

4.1.1.SPECIFICATIONS DU UNIT GLOBAL
      ----------------------------

 Le unit global contient toutes les variables globales ainsi que deux
primitives souvent utilisees: WAIT qui est une procedure d' attente
du "return" et PLACE qui renvoie vrai si il y a encore de la place dans
tabnoeud.


---------------------------------------

4.1.2.CODE
      ----*)

(*$S+*)

unit GLOBAL;
interface


const nn = 200;(* Nombre de noeuds disponibles *)
      nc = 400;(* Nombre de caracteres disponiblees *)
      nm = 4;(* Nombre de methodes *)

type typenoeud = ( pl, mu,pu,mo,val,vari );
                 (* plus, multiplie, puissance, moins, valeur, variable *)

     refnoeud = -nn..nn;(* Reference aux noeuds *)

     refcar = 1..nc;(* Reference aux caracteres *)

     noeud = record(* Noeud d'arbres *)
               tn : typenoeud;(* Type du noeud *)
               reffi : refnoeud;(* Reference au fils *)
               reffr : refnoeud;(* Refernce au frere *)
               longch : refcar;(* Longueur de la chaine de caracteres*)
               refch : refcar; (* Reference au 1er caractere de la chaine*)
```

```
                ordrinit : integer;(* Ordre initial entre freres *)
                effectue : boolean; (* Effectue au 1er degre *)
                refsol : refnoeud;(* Reference au noeud solution suivant
                                     resultat de l'application de la methode
                                     METHODE sur ce noeud*)
                methode : 0..nm;(* Methode de factorisation appliquee
                                   a ce noeud dans un arbre-solution *)
                etape : integer;(* Numero de l'etape dans le cas d'une
                                   methode a etapes *)

                case refcan : boolean of
                  false: ( ) ;
                  true : (chcan : refcar;(* Reference a la chaine de carac-
                                            teres representant l'expression
                                            canonique associee *)
                          lchcan : refcar) (* Longueur de la chaine canoni-
                                              que associee *)
                end;

     chainexp = packed array [1..82] of char;

var tabnoeud : packed array [1..nn] of noeud; (* Stack des noeuds d'arbre*)
    tabcar : packed array [1..nc] of char;(*Stack des chaines de
                                            caracteres*)
    chexp : chainexp; (* Tableau des caracteres lus a l'ecran *)
    premnoeud,dernoeud, premcar,dercar,li : integer;
    exp : refnoeud;

procedure WAIT; function PLACE : boolean;


implementation


  procedure WAIT;
    var w :char;
    begin
      writeln (' ');
      writeln ('  J ATTENDS UN RETURN  ');
      repeat read (keyboard,w) until w = chr (32)
    end;


  function PLACE;
    begin
      place := (premnoeud <= dernoeud)
    end;


begin
end.


(*---------------------------------------
```

4.1.3. TESTS

Il n'y a bien-sur aucun test sur global qui ne contient que des variables.

```
(*=======================================

                Annexe 4.2.

                UNIT LIRE

=======================================
```

4.2.1.SPECIFICATIONS

```
FUNCTION LIREXP (LI : INTEGER; VAR CHEXP : CHAINEXP) : BOOLEAN;
===============================================================
```

PRCD : LI est la ligne sur laquelle on prend les caracteres a l'ecran;
CHEXP est un tableau de caracteres .

REM : Chexp est declare en fait comme tableau a 82 caracteres plutot que 80; les 2 caracteres de plus (toujours blancs) servent a la simplification de la detection de fin de tableau car lors de la correction syntaxique, on procede "par lecture en avant" de une ou deux position; des qu'on voit un blanc , la correction syntaxique est terminee (soit fin de l'expression soit fin du tableau).

PSCD : l'expression tapee par l'utilisateur est a l'ecran a la ligne li et ne contient plus de blanc;
CHEXP est le tableau de caracteres contenant

1) des caracteres memorisables (' ','a'..'z','0'..'1','+','*', '/','^') .
2) au moins un caractere significatif (memorisables <> ' ') . Le nombre de caracteres significatifs est inferieur ou egal a 80 .
La structure de chexp est la suivante :

```
++++++++++++++++++++++++++++++++++++++++
!S !s !s !s !s !s !b !b !b !b !b !b !
++++++++++++++++++++++++++++++++++++++++
 1 ...           k  k+1      80 81 82
```

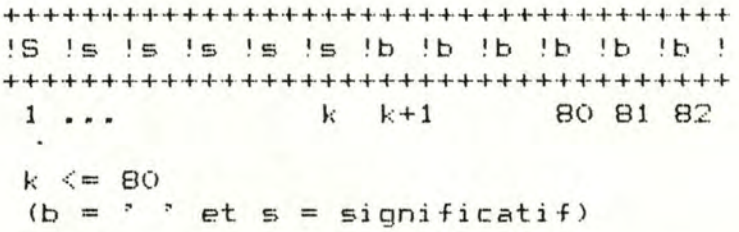
k >= 1; k <= 80
(b = ' ' et s = significatif).

renvoie VRAI si l'expression representee dans chexp est syntaxiquement correcte
FAUX sinon auquel cas le caractere '^' apparait a la ligne li+1 pour indiquer la coordonnee ou l'erreur a ete detectee et un message d'erreur apparait a la ligne li+2 .

COMMENT :

1) saisie des caracteres a l'ecran un a un
en permettant un retour en arriere du curseur
n'accepter que les caracteres 0..9,a..z,' ',+,*,/,^
le . designe la fin de la saisie a l'ecran .
(FILTRER) .

2) 'deblanchir' le tableau garni en 1)
c-a-d obtenir la structure suivante :



en profiter pour calculer le nombre de caracteres significatifs . (DEBLANCHIR) .

3) si celui ci est egal a 1, alors afficher le message correspondant et recommencer la saisie
sinon effectuer la correction syntaxique .
(SYNTAXE (chexp,pos,tmes)) .

4) si c'est correcte, alors lirexp est vrai
et un message de succes apparait a la ligne li+1
sinon lirexp est faux
l'indicateur d'erreur apparait a la ligne li+1 et
a la position de detection de l'erreur pos
et le message (TMES) correspondant apparait a l'ecran a la ligne li+2.

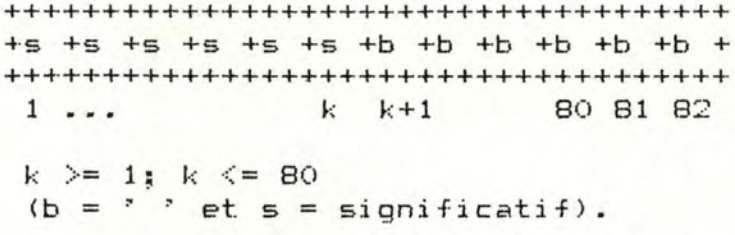
IMBRIQUES :
==========

procedures : DEBLANCHIR, FILTRER, MESSAGE

fonction : SYNTAXE .

```
FUNCTION SYNTAXE (CHEXP : CHAINEXP; VAR POS :INTEGER; VAR TMES : INTEGER)
  :BOOLEAN;
```

PRCD : CHEXP est le tableau de caracteres ne contenant que des caracteres memorisables , contenant au moins un significatif et ayant la structure suivante :



PSCD : renvoie VRAI si l'expression representee par chexp est correcte syntaxiquement auquel cas POS vaut la coordonnee du dernier caractere <> ' ' et TMES vaut 13 (message : succes de la correction)

renvoie FAUX sinon auquel cas pos = coordonnee du caractere ou l'erreur a ete detectee et TMES vaut le numero de message correspondant au type de l'erreur detectee .

SYNTAXE
=======

```
<expression> ::= <terme> ! -<terme> ! <terme> <op.add.>
                 <expression>!-<terme> <op.add.> <expression> .

<terme> ::= <facteur> ! <facteur> <op.mult.> <terme> .

<op.mult.> ::= * .

<facteur> ::= <fact.rat.> ! <fact.var.> ! <fact.exp.> .

<fact.rat.> ::= <rationnel> ! <rationnel> <op.expon.> <entier pos.> .

<fact.var.> ::= <variable> ! <variable> <op.expon.> <rationnel pos.> .

<fact.exp.> ::= (<expression>) ! (<expression>) <op.expon.>
                <entier pos.>

<op.expon.> ::= ^ .

<variable> ::= a ! ... !z .

<rationnel> ::= <entier> ! <entier> / <entier pos.> .

<rationnel pos.> ::= <entier pos.> ! <entier pos.> / <entier pos.> .

<entier> ::= 0 ! <entier pos> .

<entier pos.> ::= <chiffre non nul> ! <chiffre non nul> <entier pos> .

<chiffre nono nul> ::= 0 ! ... ! 9 .
```

REM : Le nombre de chiffres formant un entier pos. est inferieur ou egal a 3 .
Le symbole ! est equivalent a la barre verticale dans la notation backus nor form .

COMMENT :
=======

En general, construction de fonctions booleennes imbriquees . Chaque fonction sert a rechercher une certaine forme syntaxique a partir d'une position courante POS ; elle repond vrai si cette recherche reussit auquel cas POS designe la derniere position de la forme trouvee; elle repond faux sinon auquel cas POS indique la position de detection d'une erreur et TMES donne le numero du message d'erreur correspondant .

FUNCTION EXPRESSION : BOOLEAN;
==========

PRCD : POS indique la position courante dans le tableau (1<=POS<=82) a partir de laquelle il faut rechercher une expression .

PSCD : renvoie VRAI si decouverte d'une suite de caracteres dans CHEXP representant une expression de la forme :

<terme>!-<terme>!<terme><op.add><expression>!-<terme><op.add.> <expression>

renvoie FAUX sinon auquel cas pos = coordonnee du caractere ou l'erreur a ete detectee et TMES vaut le numero de message

correspondant au type de l'erreur detectee .

IMBRIQUEE :
fonction TERME .

FUNCTION TERME : BOOLEAN ;
==========

PRCD : POS indique la position courante dans le tableau
(1<=POS<=82) a partir de laquelle il faut rechercher
une terme .

PSCD : renvoie VRAI si decouverte d'une suite de caracteres dans CHEXP
representant un terme de la forme :

<facteur>!<facteur><op.mult><terme>
auquel cas pos indique la derniere position du terme
decouvert et tmes = le numeroe du message d'erreur
correspondant.

renvoie FAUX sinon auquel cas pos = coordonnee du caractere ou
l'erreur a ete detectee et TMES vaut le numero de message
correspondant au type de l'erreur detectee .

IMBRIQUEE :
fonction FACTEUR .

FUNCTION FACTEUR : BOOLEAN;
==========

PRCD : POS indique la position courante dans le tableau
(1<=POS<=82) a partir de laquelle il faut rechercher
un facteur .

PSCD : renvoie VRAI si decouverte d'une suite de caracteres dans CHEXP
represantant un facteur de la forme :
<fact.rat.>!<fact.var.>!<fact.exp.> .
auquel cas pos indique la derniere position du facteur
decouvert et tmes = le numero du message d'erreur correspondant.

renvoie FAUX sinon auquel cas pos = coordonnee du caractere ou
l'erreur a ete detectee et TMES vaut le numero de message
correspondant au type de l'erreur detectee .

IMBRIQUEE :
==========
fonction FACTRAT , FACTEXP , FACTVAR ,
RATPOS , VARIABLE, RATIONNEL , ENTIER , ENTIERPOS .

COMMENT :
=======
1.La decoupe de FACTEUR en FACTRAT, FACTEXP et FACTVAR simplifie la
comprehention de ce module .

2.Une variable FIN booleenne est utilisee pour savoir si ,a la suite
de l'application d'une sous fonction (factvar, ...), il faut en essayer
une autre . En realite , chaque sous-fonction peut conduire a une des
3 conclusions suivantes : ex factvar

2.1 fin = vrai , factvar = faux (erreur dans l'exposant d'une puissance; il ne faut pas essayer une autre sous-fonction))
2.2 fin = faux , factvar = faux (on ne trouve pas de variable; il faut essayer une autre sous-fonction)
2.3 factvar = vrai

```
FUNCTION FACTRAT : BOOLEAN ;
===========================
```

PRCD : POS indique la position courante dans le tableau (1<=POS<=82) a partir de laquelle il faut rechercher un fac.rat. .

PSCD : renvoie VRAI si decouverte d'une suite de caracteres dans CHEXP representant un fac.rat. de la forme : <rationnel>!<rationnel><op.expon.><rationnel pos.> .. auquel cas pos indique la derniere position du fact.rat decouvert et tmes = le numero du message d'erreur correspondant.

renvoie FAUX sinon auquel cas pos = coordonnee du caractere ou l'erreur a ete detectee et TMES vaut le numero de message correspondant au type de l'erreur detectee .

```
FUNCTION FACTVAR : BOOLEAN ;
============================
```

PRCD : POS indique la position courante dans le tableau (1<=POS<=82) a partir de laquelle il faut rechercher un fac.var. .

PSCD : renvoie VRAI si decouverte d'une suite de caracteres dans CHEXP representant un fac.var. de la forme : <variable>!<variable><op.expon.><entier pos.> .. auquel cas pos indique la derniere position du fact.var decouvert et tmes = le numero du message d'erreur correspondant.

renvoie FAUX sinon auquel cas pos = coordonnee du caractere ou l'erreur a ete detectee et TMES vaut le numero de message correspondant au type de l'erreur detectee .

```
FUNCTION FACTEXP : BOOLEAN ;
============================
```

PRCD : POS indique la position courante dans le tableau (1<=POS<=82) a partir de laquelle il faut rechercher un fac.exp. .

PSCD : renvoie VRAI si decouverte d'une suite de caracteres dans CHEXP representant un fac.exp. de la forme : (<expression>)!(<expression>)<op.expon.><entier pos.> .. auquel cas pos indique la derniere position du fact.exp decouvert et tmes = le numero du message d'erreur correspondant.

renvoie FAUX sinon auquel cas pos = coordonnee du caractere ou l'erreur a ete detectee et TMES vaut le numero de message

correspondant au type de l'erreur detectee .

FUNCTION RATPOS : BOOLEAN ;
==========

PRCD : POS indique la position courante dans le tableau
(1<=POS<=82) a partir de laquelle il faut rechercher
un rationnel positif .

PSCD : renvoie VRAI si decouverte d'une suite de caracteres dans CHEXP
representant un rationnel positif de la forme :
<entier pos>!<entier pos> / <entier pos.> .
auquel cas POS indique la derniere position du rationnel positif
decouvert et tmes = le numero du message d'erreur correspondant.

renvoie FAUX sinon auquel cas pos = coordonnee du caractere ou
l'erreur a ete detectee et TMES vaut le numero de message
correspondant au type de l'erreur detectee .

FUNCTION VARIABLE BOOLEAN ;
==========

PRCD : POS indique la position courante dans le tableau
(1<=POS<=82) a partir de laquelle il faut rechercher
une variable .

PSCD : renvoie VRAI si decouverte d'une suite de caracteres dans CHEXP
representant une variable de la forme :
a ! ... !z .
auquel cas POS indique la derniere position de la variable
decouverte et tmes = le numero du message d'erreur
correspondant.

renvoie FAUX sinon auquel cas pos = coordonnee du caractere ou
l'erreur a ete detectee et TMES vaut le numero de message
correspondant au type de l'erreur detectee .

FUNCTION RATIONNEL BOOLEAN ;
==========

PRCD : POS indique la position courante dans le tableau
(1<=POS<=82) a partir de laquelle il faut rechercher
un rationnel .

PSCD : renvoie VRAI si decouverte d'une suite de caracteres dans CHEXP
representant un rationnel de la forme :
<entier>!<entier> / <entier pos.>
auquel cas POS indique la derniere position du rationnel
decouverte et tmes = le numero du message d'erreur
correspondant.

renvoie FAUX sinon auquel cas pos = coordonnee du caractere ou
l'erreur a ete detectee et TMES vaut le numero de message
correspondant au type de l'erreur detectee .

```
FUNCTION ENTIER BOOLEAN ;
=========================

  PRCD : POS indique la position courante dans le tableau
              (1<=POS<=82) a partir de laquelle il faut rechercher
              un entier .

  PSCD : renvoie VRAI si decouverte d'une suite de caracteres dans CHEXP
           representant un entier de la forme :
           0 !<entier pos.>
           auquel cas POS indique la derniere position de l'entier
           decouvert et tmes = le numero du message d'erreur correspondant.

         renvoie FAUX sinon auquel cas pos = coordonnee du caractere ou
           l'erreur a ete detectee et TMES vaut le numero de message
           correspondant au type de l'erreur detectee .

           ---------------------------------

FUNCTION ENTIERPOS BOOLEAN ;
============================

  PRCD : POS indique la position courante dans le tableau
              (1<=POS<=82) a partir de laquelle il faut rechercher
              un entier positif .

  PSCD : renvoie VRAI si decouverte d'une suite de caracteres dans CHEXP
           representant un entier positif de la forme :
           <chiffre non nul> ! <chiffre non nul> <entier pos.> (nbre de
           chiffre <= 3)
           auquel cas POS indique la derniere position de l'entier positif
           decouvert et tmes = le numero du message d'erreur correspondant.

         renvoie FAUX sinon auquel cas pos = coordonnee du caractere ou
           l'erreur a ete detectee et TMES vaut le numero de message
           correspondant au type de l'erreur detectee .

-------------------------------------*)
```

```
(*
4.2.2.CODE
      ----*)


(*$S+*)

unit LIRE;

interface


  uses (*$U #5:GLOBAL.CODE*) global;
  function LIREXP (li : integer; var chexp : chainexp) : boolean;

implementation

function LIREXP ;


  var x,compt ,i,k,pos,tmes :integer;
      fin,ok : boolean;


  function ACCEPTABLE (car : char) : boolean;
    begin
      acceptable := ((car in
        ['A'..'Z','0'..'9','+','-','*','^','(',')',' ','.','/'])
        or (ord (car) = 8) or (ord (car) = 6))
    end;


  function MEMORISABLE (car :char) : boolean;
    begin
      memorisable :=
      (car in ['A'..'Z','0'..'9','+','-','*','^','(',')',' ','/'])
    end;


  procedure MESSAGE (tmes : integer); (* pas encore au point *)
    begin
      case tmes of
        0 : writeln ('  J ATTENDS AU MOINS UN CARACTERE  ');
        1 : writeln ('  J ATTENDS AU MOINS UN CARACTERE SIGNIFICATIF  ');
        2 : writeln ('  J AI 80 CARACTERES  ');
        3 : writeln ('  J ATTENDS UNE EXPRESSION  ');
        5 : writeln ('  J ATTENDS UN TERME  ');
        6 : writeln ('  J ATTENDS UN FACTEUR  ');
        8 : writeln ('  ENTIER TROP LONG : 3 CHIFFRES MAX  ');
        9 : writeln ('  J ATTENDS UNE )  ');
        10: writeln ('  J ATTENDS UNE ( OU UN ENTIER OU UNE VARIABLE'  );
        11: writeln ('  J ATTENDS UN ENTIER POUR UNE PUISSANCE  ');
        12: writeln ('  THAT S ALL LOLKS !  ');
        13: writeln ('  BONNE EXPRESSION JUAQU AU ^  ');
        14: writeln ('  YOU ARE A GOOD BOY !  ');
        15: writeln ('  IDENTIFICATEUR DE VARIABLE TROP LONG  ');
        16: writeln ('  J ATTENDS UNE VARIABLE  ');
        17: writeln ('  J ATTENDS UN RATIONNEL  ');
```

```
      18: writeln ('  J ATTENDS UN ENTIER  ');
      19: writeln ('  ZERO MAL PLACE  ');
      20: writeln ('  LE QUOTIENT N EXISTE PAS  ');
      21: writeln ('  J ATTENDS UN ENTIER POSITIF  ');
      22: writeln ('  J ATTENDS UN CHIFFRE NON NUL  ');

    end
  end;

procedure FILTRER;

  var ch : char;

  begin (*FILTRER*)
    repeat read (keyboard,ch) until acceptable(ch);
    if memorisable(ch) then
      begin
        compt := compt + 1;
        chexp [compt] := ch;
        write (ch);x := x + 1;
        if compt >= 80 then fin := true
      end
    else
      begin
        case ord (ch) of
          46 : fin := true; (* '.' = fin *)
          8  : begin (* fleche d'effacement vers la gauche *)
                 if compt >= 1 then
                   begin
                     chexp [compt] := chr (32);
                     compt := compt - 1;
                     x := x - 1;gotoxy (x,1i);write (' ');gotoxy (x,1i)
                   end

               end
        end
      end
  end;(*FILTRER*)

procedure DEBLANCHIR;
  begin
    k := 0;
    for i := 1 to compt do
      begin
        if chexp [i] <> chr(32) then
          begin
            k := k + 1;
            if k < i then
              begin
                chexp [k] := chexp [i];
                chexp [i] := chr(32)
              end
          end
      end
  end;
```

```
function SYNTAXE (chexp : chainexp; var pos : integer; var tmes : integer)
                  boolean;

  function EXPRESSION : boolean;

    function TERME : boolean;

      function FACTEUR : boolean;

      var fin : boolean;

      function ENTIERPOS : boolean;

        var p : integer;

        begin (* entierpos *)
          if chexp [pos] = chr (32) then
            begin
              entierpos := false; tmes := 21
            end
          else
            if chexp [pos] in ['1'..'9'] then
              begin
                p := pos;
                repeat p := p + 1 until not (chexp [p] in ['0'..'9']);
                if (p - pos) >= 4 then
                  begin
                    entierpos := false; tmes := 8; fin := true
                  end
                else entierpos := true;
                pos := (p - 1)
              end
            else
              begin
                entierpos := false; tmes := 22
              end;
        end; (* entierpos *)

      function ENTIER : boolean;

        begin (* entier *)
          if chexp [pos] = chr (32) then
            begin
              entier := false; tmes := 18
            end
          else
            if chexp [pos] = '0' then
              begin
                if chexp [pos + 1] in ['0'.. '9'] then
                  begin
                    entier := false; tmes := 19; pos := pos +1;
                    fin := true
                  end
```

```
          else entier := true
        end
      else
        if entierpos then entier := true
        else entier := false;
  end; (* entier *)


function RATIONNEL : boolean;


  begin (* rationnel *)
    if chexp [pos] = chr(32) then
      begin
        rationnel := false; tmes := 17
      end
    else
      if entier then
        if chexp [pos + 1] = '/' then
          begin
            pos := pos + 2;
            if entierpos then rationnel := true
            else
              begin
                fin := true;
                rationnel := false
              end
          end
        else rationnel := true
      else rationnel := false;
  end; (* rationnel *)


function VARIABLE : boolean;


  begin (* variable *)
    if chexp [pos] = chr (32) then
      begin
        variable:= false; tmes := 16
      end
    else
      if chexp [pos]  in [ 'A'..'Z'] then
        if chexp [pos + 1] in ['A'..'Z'] then
          begin
            pos := pos + 1;
            variable := false; tmes := 15; fin := true;fin := true
          end
        else variable := true
      else
        begin
          variable := false; tmes := 16
        end;
  end; (*variable*)
```

```
function FACTEXP : boolean;


  begin (* factexp *)
    if chexp [pos] = chr (32) then
      begin
        factexp := false; tmes := 28; fin := true
      end
    else
      if chexp [pos] = '(' then
        begin
          pos := pos + 1;
          if expression then
            if chexp [pos + 1] = ')' then
              begin
                pos := pos + 1;
                if chexp [pos + 1] = '^' then
                  begin
                    pos := pos + 2;
                    if entierpos then factexp := true
                    else
                      begin
                        factexp := false; fin := true; tmes := 11
                      end
                  end
                else factexp := true
              end
            else
              begin
                factexp := false; tmes := 9; fin := true;
                pos := pos + 1
              end
          else
            begin
              factexp := false; fin := true
            end
        end
      else
        begin
          factexp := false; tmes := 30
        end;
    end; (* factexp *)


  function RATPOS : boolean;


    begin (* ratpos *)
      if chexp [pos] = chr (32) then
        begin
          ratpos := false; tmes := 20
        end
      else
        if entierpos then
          begin
            if chexp [pos + 1] = '/' then
              begin
                pos := pos + 2;
                if entierpos then ratpos := true
```

```
                else
                  begin
                    ratpos := false; fin := true
                  end
              end
            else ratpos := true
          end
        else ratpos := false;
    end; (* ratpos *)

function  FACTRAT : boolean;

    begin (* factrat *)
      if chexp [pos] = chr (32) then
        begin
          factrat := false; tmes := 29; fin := true
        end
      else
        begin
          if rationnel then
            if chexp [pos + 1] = '^' then
              begin
                pos := pos + 2;
                if ratpos then factrat := true
                else
                  begin
                    factrat := false; fin := true
                  end
              end
            else factrat := true
          else factrat := false
        end;
    end; (* factrat *)

function FACTVAR : boolean;


    begin (* factvar *)
      if chexp [pos] = chr (32) then
        begin
          factvar := false; tmes := 31; fin := true
        end
      else
        if variable then
          if chexp [pos + 1] = '^' then
            begin
              pos := pos + 2;
              if not (chexp [pos] in ['0'..'9']) then
                begin
                  factvar := false; fin := true; tmes := 11
                end
              else
                if entierpos then factvar := true
              else
                begin
                  factvar := false; fin := true
                end
            end
```

```
          else
            if chexp [pos + 1] = '/' then
              begin
                factvar := false; pos := pos + 1; fin := true;
                tmes := 20
              end
            else factvar := true
        else factvar := false;
    end; (* factvar *)


  begin (* facteur *)
    fin := false;
    if chexp [pos] = chr (32) then
      begin
        facteur := false; tmes := 6
      end
    else
      if factvar then facteur := true
      else
        if not fin then
          if factrat then facteur := true
          else
            if not fin then
              if factexp then facteur := true
              else
                if not fin then
                  begin
                    facteur := false;
                    tmes := 6
                  end
            else facteur := false

        else facteur := false;
  end; (* facteur *)



begin (* terme *)
  if chexp [pos] = chr (32)then
    begin
      terme := false; tmes := 5
    end
  else
    if facteur then
      if chexp [pos + 1] = '*' then
        begin
          pos := pos + 2;
          terme := terme;
        end
      else terme := true
    else terme := false;
 end; (* terme *)




begin (* expression *)
  if chexp [pos] = chr (32) then
    begin
      expression := false; tmes := 3
```

```
      end
    else
      if chexp [pos] = '-' then
        begin
          pos := pos + 1;

          if chexp [pos] = chr (32) then
            begin
              expression := false; tmes := 5
            end
          else
            if terme then
              if chexp [pos + 1] in ['+','-'] then
                begin
                  pos := pos + 2; expression := expression
                end
              else expression := true
            else expression := false
        end
      else
        if terme then
          if chexp [pos + 1] in ['+','-'] then
            begin
              pos := pos + 2;
              expression := expression
            end
          else expression := true
        else expression := false;
    end; (* expression *)


begin (* syntaxe *)
  pos := 1;

  if expression then
   if chexp [pos + 1] = chr (32) then syntaxe := true
   else
     begin
       syntaxe := false; tmes := 13; pos := pos + 1
     end
  else syntaxe := false
end; (* syntaxe *)


begin(*lirexp*)
  repeat

    gotoxy (0,1i); gotoxy (80,1i); gotoxy (0,1i);
    for i := 1 to 82 do chexp [i] := chr (32);
    x := 0;compt := 0; fin := false;

    repeat filtrer until fin = true;

    if compt = 0 then begin
                        gotoxy (0,1i + 1);message (0); lirexp := false;
                        wait;
                        page (output)
                      end
  until compt > 0;

  if compt = 80 then
    begin
```

```
        gotoxy (0,li + 1);message (2);wait;page (output)
      end;
    deblanchir;
    if k = 0 then
      begin
        gotoxy (0,li + 2);
        message (1); lirexp := false
      end
    else
      begin
        gotoxy (0,0);
        gotoxy (0,1);
        ok := syntaxe (chexp,pos,tmes);
        page (output);
        for i := 1 to compt do write (chexp [i]);
        if not ok then
          begin
            gotoxy (pos - 1,li + 1);
            writeln ('^');
            gotoxy (0,li + 2);
            message (tmes);
            gotoxy (0,li + 3);
            lirexp := false
          end
        else
          begin
            gotoxy (0,li + 1);
            message (14);
            gotoxy (0,li + 3);
            lirexp := true
          end;
        wait
      end
  end;(*lirexp*)



begin
end.

(*-----------------------------------*)
```

```
(*
4.2.3.TEST
      ----*)

(*$S+*)

program TESTLIRE;


uses (*$U #5:GLOBAL.CODE*) global,(*$U #5:LIRE.CODE*) lire;

  var veux : boolean;
      p : char;
      po,i : integer;


begin (*SYNT*)
  veux := true;
  repeat
    begin
      page (output);
      gotoxy (0,0);

      write ('  DONNEZ MOI UNE POSITION A L ECRAN : <0..9)  ');
      repeat read (keyboard,p) until p in ['0'..'9'];
      writeln (p);
      page (output);
      case p of
        '0' : po := 0;
        '1' : po := 1;
        '2' : po := 2;
        '3' : po := 3;
        '4' : po := 4;
        '5' : po := 5;
        '6' : po := 6;
        '7' : po := 7;
        '8' : po := 8;
        '9' : po := 9
      end;

      writeln ('  TAPE YOUR EXPRESSION  ');

      veux := lirexp (po,chexp);

      write ('  NOG  ? <O,N>  ');
      repeat read (keyboard,p) until p in ['O','N'];
      writeln (p);
      if p = 'N' then veux := false else veux := true
    end
  until veux = false

end.

(*---------------------------------------
---------------------------------------*)
```

```
(*=======================================
```

Annexe 4.3.

UNIT SUNBROTHER

```
=========================================
```

4.3.1.SPECIFICATIONS

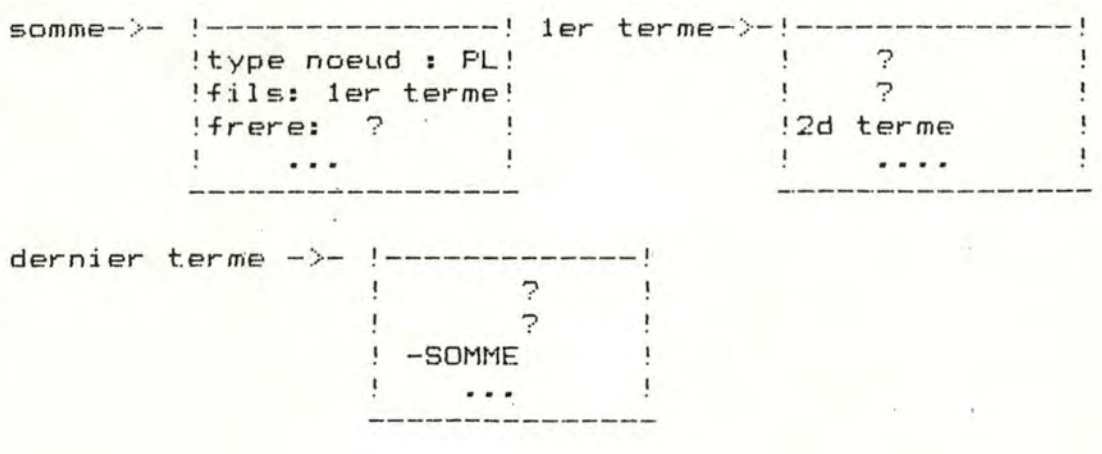
DESCRIPTION DE L'ARBRE FILS-FRERE

Chaque noeud d'arbre est un record appartenant a un tableau .

Chaque noeud contient :
- son type : plus,multiplie,puissance,moins,valeur,variable
- ses references : - au fils
  - au frere
- sa description en chaine de caracteres : - reference du premier caractere dans le stack des caracteres
  - longueur de la chaine (nbre de caracteres)
- sa description dans un arbre solution : - reference au noeud suivant dans l'arbre solution
  - la methode
  - l'etape (v.construction d'une solution)
- des 'utilItitaires' : - effectue ou non
  - references a l'expression canonique associee

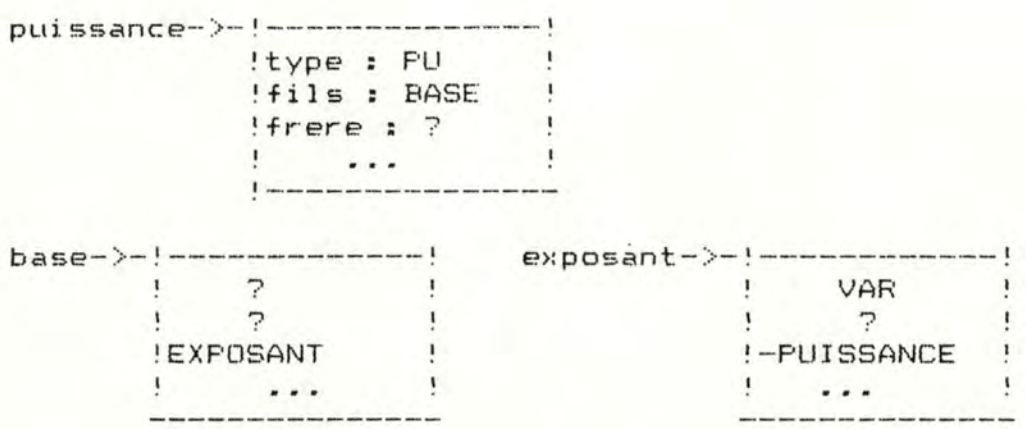
Dans la construction de l'arbre fils-frere, chaque champs sera tel que

-un noeud racine : reference au frere = 0
-un noeud somme :

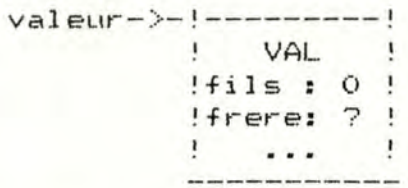


-un noeud produit : meme structure avec le type-pere = mu

-un noeud puissance :



-un noeud valeur :



-un noeud variable : meme structure avec type = VARI

IMPORTANTS

1.Toute demande de place (creer un noeud) dans les stacks (stack-noeud et stack-caracteres) ayant essuie un refus coupe court au traitement demandeur .
Ainsi, si il n'y a pas de place pour creer le second terme d'une somme alors la construction de l'arbre est stoppee et un message d'erreur apparait .

2.Etapes de la construction d'un arbre EXPRESSION :

2.1 Lirexp : saisie a l'ecran et correction syntaxique

2.2 Filsfrere : construction de l'arbre fils-frere a l'ordre pres

2.3 Ordonner : construction de l'arbre expression = l'arbre fils-frere ordonne .

3.Les parentheses restent chez les enfants :
exemple :
pere (x-2)^3
fils (x-2) et frere du fils 3

4.Les champs ordrinit (ordre initial) sont garni dans l'ordre ou les operandes (enfants) se presentent .

FUNCTION FILSFRERE (CHEXP : CHAINEXP; VAR EXP : REFNOEUD) : BOOLEAN;
==========

PRCD : Il y a encore de la place dans les deux stacks .
Chexp est un tableau de 82 caracteres tel que
2.1 l'expression representee par ce tableau est correcte syntaxiquement (v.lirexp)
2.2 la structure de chexp :
chexp ne contient - que des caracteres significatifs
- au plus 80 caracteres significatifs

```
          ----------------------------
chexp :   !s!s!s!s!s!s!s!s!b! ... !b!b!
          ----------------------------
           1 2  ...        k       8182   k >= 1
```

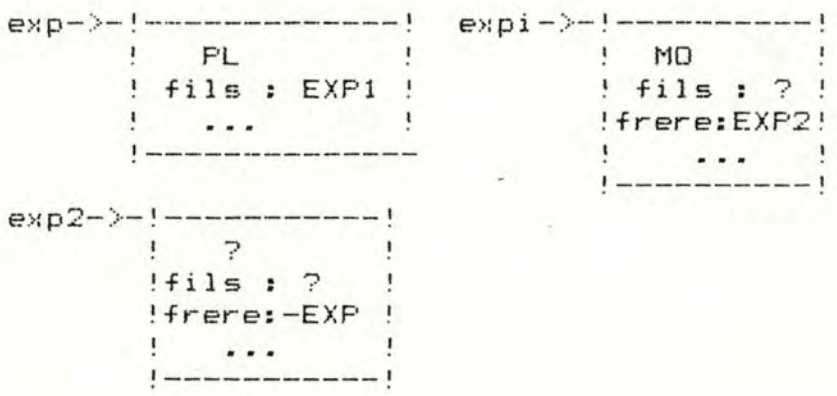
PSCD : Renvoie VRAI si il y a eu assez de place dans les stacks pour construire l'arbre fils-frere correspondant a l'expression representee par la chaine de caractere dans chexp

renvoie FAUX sinon auquel cas exp = 0 .

REM NOTATION :

1.Par soucis de clarte et de concision, une expression sera notee par <EXP> ou exp designe la coordonne dans tabnoeud (stack-noeud) de la racine de l'arbre representant cette expression .

2.<EXP> = -<EXP1> + <EXP2> exprime le fait que l'expression <exp> est la somme du 'moins monadique' -<exp1> et de <exp2> .



COMMENT

Des fonctions booleennes calquant la syntaxe d'une expression : expression-terme-facteur-valeur (rationnel,entier,...) -variable renvoyant vrai si il y a eu suffisament de place pour construire le sous-arbre fils-frere correspondant .

Chaque fonction aura 3 parametres :
-debut : situe dans chexp le debut de la sous-expression consideree
-fin : situera la fin dans chexp de la sous-expression cosideree
-exp : donnera la coordonnee dans le stack-noeud (tabnoeud) de la racine de la sous-expression consideree .

IMBRIQUEES :

Expression .

```
FUNCTION EXPRESSION (DEBUT : INTEGER; VAR FIN : INTEGER; VAR EXP : REFNOEUD
                    ): BOOLEAN
```

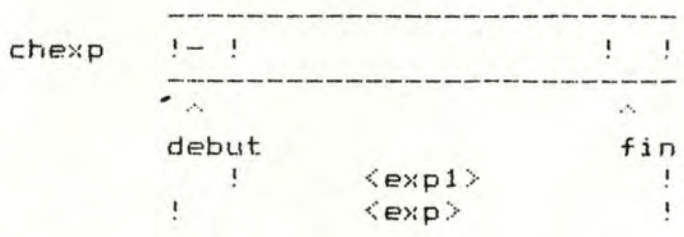
PRCD : 1 <= debut <= 80; chexp est le meme tableau que decrit dans la prcd de FILSFRERE .

PSCD : Renvoie VRAI si il ya eu assez de place dans les stacks pour construire l'arbre fils-frere representant l'expression dont le 1er caractere se situe en chexp [debut]
auquel cas exp est la coordonnee dans tabnoeud de la racine de cet arbre; fin indique la coordonnee dans chexp du dernier caractere de cette expression

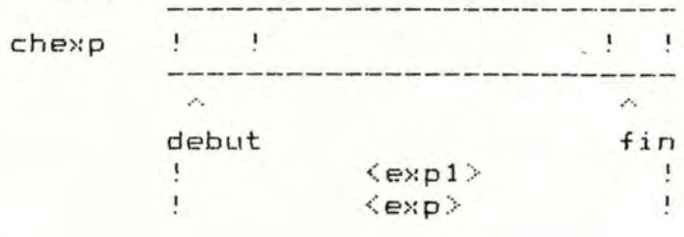
renvoie FAUX sinon auquel cas exp = 0 .

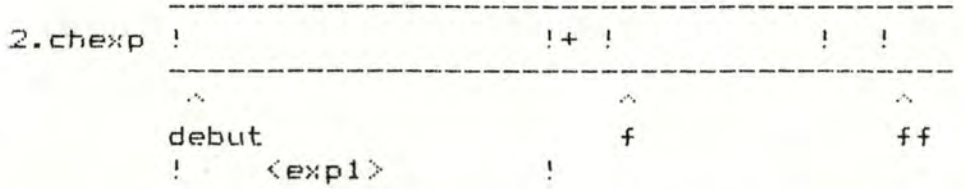
COMMENT : Cas a traiter :

1. <exp> = <exp1> ou exp1 est un terme



<exp> = - <exp1>, l' expression en question est un moins monadique





1.Donc, on recherche d'abord un premier terme (moinmona ou pas). Si cette recherche echoue, expression echoue; sinon, si il n' y a pas de signe "+" ou "-" a la suite du premier terme, ce n' est pas une somme, sinon 2.

2.Tant qu'il y a un signe "+" ou "-" a la suite du dernier terme decouvert, rechercher le terme suivant. Si une de ces recherche echoue, expression echoue. A chaque terme decouvert, assurer leur enchainement par le champs reffr. Expression vaudra vrai si il ya suffisamment de place pour construire le noeud pere du produit en question.

IMBRIQUES :

TERME, MOINSMONA, TRTSOM.

FUNCTION TESTPLACE : BOOLEAN;

PRCD : 1 <= ff et debut <= 80

PSCD : Renvoie expression a vrai s'il y a assez de place dans les stacks pour reserver un noeud racine et pour reserver des caracteres en suffisance pour le champs 'chaine de carac.'

renvoie expression a faux et exp a 0 sinon .

FUNCTION TERME (DEBUT : INTEGER; VAR FIN : INTEGER; VAR EXP : REFNOEUD): BOOLEAN

PRCD : 1 <= debut <= 80; chexp est le meme tableau que decrit dans la prcd de FILSFRERE .

PRCD : Renvoie VRAI si il ya eu assez de place dans les stacks pour construire l'arbre fils-frere representant le terme dont le 1er caractere se situe en chexp [debut]
auquel cas exp est la coordonnee dans tabnoeud de la racine de cet arbre; fin indique la coordonnee dans chexp du dernier caractere de ce terme

renvoie FAUX sinon auquel cas exp = 0 .

COMMENT :

1.Rechercher d'abord un premier facteur. Si la recherche echoue, terme echoue; sinon 2.

2.Tant qu'il y a un signe '*' a la suite du dernier facteur decouvert, rechercher le facteur suivant. Si une recherche echoue, terme echoue. A chaque facteur decouvert, assurer leur enchainement par les champs reffr. Lorsque tous les facteurs sont decouverts sans aucun echec, construire le noeud pere (exp) du produit.

IMBRIQUE :

FACTEUR, CREERPROD.

PROCEDURE CREERPROD;

PRCD: exp1 et derf sont respectivement le premier et le dernier facteur d'un produit qui commence en debut et qui termine en fin.

PSCD: renvoie terme a vrai si il y a assez de place pour creer le noeud pere du produit <exp> de premier facteur <exp1> et de dernie

```
facteur <derf>
 renvoie terme a faux et exp a 0 sinon.

-------------------------*)
```

```
(*
FUNCTION FACTEUR (DEBUT;VAR FIN : INTEGER; VAR EXP : REFNOEUD) : BOOLEAN
-------------------------------------------------------------------------

  PRCD : 1 <= debut <= 80; chexp est le meme tableau que decrit dans la
         prcd de FILSFRERE .

  PRCD : Renvoie VRAI si il ya eu assez de place dans les stacks pour cons-
         truire l'arbre fils-frere representant le facteur dont le 1er
         caractere se situe en chexp [debut]
         auquel cas exp est la coordonnee dans tabnoeud de la racine de cet
         arbre; fin indique la coordonnee dans chexp du dernier caractere
         de ce facteur

          renvoie FAUX sinon auquel cas exp = 0 .

  COMMENT :

    3 cas :  ( <expression> ) ou  ( <expression> ) ^ <valeur>
             <variable> ou <variable> ^ <valeur>
             <valeur> ou <valeur> ^ <valeur>

    qui correspondent aux 3 traitements : trtexp
                                          trtvar
                                          trtrat


  IMBRIQUE :

    TRTEXP, TRTVAR, TRTRAT, PUISSANCE, VARIABLE, VALEUR .

          --------------------

FUNCTION VALEUR (DEBUT : INTEGER; VAR FIN : INTEGER; VAR EXP : REFNOEUD
                 ) : BOOLEAN
-------------------------------------------------------------------------

  PRCD : 1 <= DEBUT <= 82;
         CHEXP [DEBUT] not in ['a'..'z'];
         CHEXP [DEBUT] <> '(';

  PSCD : Renvoie VRAI si il ya eu assez de place pour construire
         le sous-arbre fils-frere representant la valeur qui debute en
         chexp [debut]; exp est la coordonnee dans tabnoeud du noeud qui
         represente cette valeur; fin est la coordonnee dans chexp du
         dernier caractere de cette valeur

          renvoie FAUX sinon auquel cas exp = 0 .

          --------------------

FUNCTION VARIABLE (DEBUT : INTEGER; VAR FIN : INTEGER; VAR EXP : REFNOEUD
                 ) : BOOLEAN
-------------------------------------------------------------------------

  PRCD : 1 <= DEBUT <= 82;
         CHEXP [DEBUT] in ['a'..'z'];

  PSCD : Renvoie VRAI si il ya eu assez de place pour construire
         le sous-arbre fils-frere representant la variable qui trouve en
         chexp [debut]; exp est la coordonnee dans tabnoeud du noeud qui
```

represente cette variable; fin est la coordonnee dans chexp du dernier caractere de cette variable

renvoie FAUX sinon auquel cas exp = 0 .

PROCEDURE TRTEXP ;

PRCD : 1 <= DEBUT <= 82;
CHEXP [DEBUT] = '(';

PSCD : Renvoie facteur a vrai si il ya eu assez de place pour construire le sous-arbre fils-frere representant soit l'expression qui commence en chexp [debut] (<exp> = (<exp1>)) soit la puissance commencant exp est la coordonnee dans tabnoeud de la racine du sous-arbre construit; fin est la coordonnee dans chexp du dernier caractere de cette sous-expression

---

PROCEDURE TRTVAR;

PRCD : 1 <= DEBUT <= 82;
CHEXP [DEBUT] in ['a'..'z'];

PSCD : Renvoie facteur a vrai si il ya eu assez de place pour construire le sous-arbre fils-frere representant soit la variable qui commence en chexp [debut] soit la puissance commencant en chexp [debut] (<exp> = <exp1> ^ <exp2>) ; dans les deux cas, exp est la coordonnee dans tabnoeud de la racine du sous-arbre construit; fin est la coordonnee dans chexp du dernier caractere de cette sous-expression

renvoie facteur a FAUX sinon auquel cas exp = 0 .

PROCEDURE TRTRAT;

PRCD : 1 <= DEBUT <= 82;
CHEXP [DEBUT] not in ['a'..'z'];
CHEXP [DEBUT] <> '(';

PSCD : Renvoie facteur a vrai si il ya eu assez de place pour construire le sous-arbre fils-frere representant soit la valeur qui commence en chexp [debut] soit la puissance commencant en chexp [debut] (<exp> = <exp1> ^ <exp2>) ; dans les deux cas, exp est la coordonnee dans tabnoeud de la racine du sous-arbre construit; fin est la coordonnee dans chexp du dernier caractere de cette sous-expression

renvoie facteur a FAUX sinon auquel cas exp = 0 .

REM : Valeur represente tout ce qui est rationnel, entier, positif ou non .
Le '/' n'est pas considere comme un operateur .

PROCEDURE PUISSANCE (EXP1,EXP2);

PRCD :

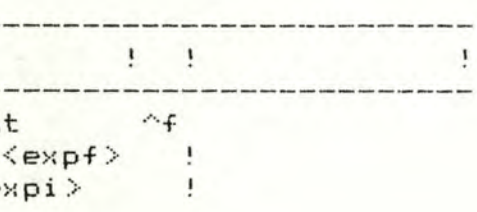
PSCD : Renvoie facteur a FAUX s'il y a assez de place pour construire le sous-arbre fils-frere puissance (<exp> = <exp1> ^ <exp2>); auquel cas exp designe la coordonnee dans tabnoeud de la racine du sous-arbre construit; fin designe la coorddonnee dans tabcar du dernier caractere de cette sous-expression

renvoie facteur a FAUX sinon auquel cas exp = 0 .

FUNCTION MOINSMONA : BOOLEAN

PRCD : chexp [debut] = '=';
terme (debut+1,f,expf) = true;

illustration :



PSCD : Renvoie VRAI si il y a eu suffisament de place dans les stacks pour creer la racine de l'arbre fils-frere <expi> 'moins monadique' de fils <expf> (<expi> = - <expf>;

renvoie FAUX sinon .

PROCEDURE TRTSOM (PREMT: REFNOEUD)

PRCD : <premt> est le premier terme d' une somme.

PSCD : renvoie expression a vrai si il ya eu assez de place pour construire la somme de 1er terme <expt> ; auquel cas exp est la coordonnee dans tabnoeud de la racine de l'arbre somme construit et fin est la position du dernier caractere de celle-ci.

renvoie faux exp a 0 sinon .

PROCEDURE CREERSOM;

PRCD : <premt> et <dert> sont, respectivement, le premier et dernier

terme d'une somme dont tous les termes sont deja construits.

PSCD : renvoie expression a vrai si il y a suffisammnent de place pour construire le noeud pere <exp> de la somme dont le premier terme est premt et le dernier dert
renvoie expression a faux et exp a 0 sinon.

----------*)

```
(*
4.3.2.CODE
      -----*)


(*$S+*)
unit SUNBROTHER;

interface

  uses (*$U #5:GLOBAL.CODE*) global;
  function FILSFRERE (chexp : chainexp;var exp : refnoeud) : boolean;

implementation

function FILSFRERE ;

(* Construction de l' arbre fils-frere a partir du tableau de
caracteres chexp *)

  var f,i,j,k : integer;


  function EXPRESSION (debut : integer; var fin : integer; var exp :
                       refnoeud) : boolean;

    var f : integer;
        exp1, expi : refnoeud;


  function MOINSMONA (expf : refnoeud;var expi : refnoeud) : boolean;


    begin (* Moinsmona *)

      (* Moinsmona : realiser un arbre filsfrere (<EXPI>) 'moins' de
              racine expi et de fils expf (<EXPI> = -<EXPF>)*)

        if not place then moinsmona := false
        else
          begin
            moinsmona := true;
            expi := premnoeud; premnoeud := premnoeud + 1;
            tabnoeud[expi].tn := mo;
            tabnoeud[expi].reffi := expf;
            tabnoeud[expi].reffr := 0;
            tabnoeud[expi].longch := tabnoeud [expf].longch + 1;
            tabnoeud[expi].refch := tabnoeud [expf].refch - 1;
            tabnoeud[expi].ordrinit := 1;
            tabnoeud[expi].effectue := false;
            tabnoeud[expi].refsol := 0;
            tabnoeud[expi].methode := 0;
            tabnoeud[expi].etape := 0;
            tabnoeud[expi].refcan := false;

            tabnoeud[expf].reffr := -expi
          end;
    end; (* Moinsmona *)
```

```
function TERME (debut :integer; var fin :integer; var exp : refnoeud
                ) : boolean;

  var f,fc,dc : integer;
      exp1,derf,fcrt : refnoeud;
      termine : boolean;

  procedure CREERPROD;


    begin (* Creeprod *)
      if not place then
        begin
          terme := false; exp := 0
        end
      else
        begin
          exp := premnoeud; premnoeud := premnoeud + 1;
          fin := fc;
          tabnoeud[exp].tn := mu;
          tabnoeud[exp].reffi := exp1;
          tabnoeud[exp].reffr := 0;
          tabnoeud[exp].longch := (fin - debut + 1);
          tabnoeud[exp].refch := debut;
          tabnoeud[exp].ordrinit := 1;
          tabnoeud[exp].effectue := false;
          tabnoeud[exp].refsol := 0;
          tabnoeud[exp].methode := 0;
          tabnoeud[exp].etape := 0;
          tabnoeud[exp].refcan := false;

          tabnoeud [derf].reffr := -exp;
        end
  end; (* Creerprod *)


function FACTEUR (debut : integer; var fin : integer; var exp : refnoeud
                  ) : boolean;
  var exp1,exp2 : refnoeud;
      f,ff : integer;

function VALEUR (debut : integer; var fin : integer; var exp : refnoeud
                 ) : boolean;


  begin (* Valeur *)
    if not place then
      begin
        valeur := false; exp := 0
      end
    else
      begin
        fin := debut;
        while ((tabcar [fin] in (['0'..'9'] + ['/'])) and
               (fin <= (j+k-1)))
            do fin := fin + 1; (* j = debut de l'expression en entier;
                                  k = la longueur de l'expression en entier
                                  *)
        fin := fin - 1;
```

```
        valeur := true;
        exp := premnoeud;premnoeud := premnoeud + 1;
        tabnoeud[exp].tn := val;
        tabnoeud[exp].reffi := 0;
        tabnoeud[exp].reffr := 0;
        tabnoeud[exp].longch := (fin - debut + 1);
        tabnoeud[exp].refch := debut;
        tabnoeud[exp].ordrinit := 1;
        tabnoeud[exp].effectue := false;
        tabnoeud[exp].refsol := 0;
        tabnoeud[exp].methode := 0;
        tabnoeud[exp].etape := 0;
        tabnoeud[exp].refcan := false;
      end
  end; (* Valeur *)

function VARIABLE (debut : integer; var fin : integer; var exp : refnoeud
                  ) : boolean;

  begin (* Variable *)
      if not place then
        begin
          variable := false; exp := 0
        end
      else
        begin
          fin := debut;
          variable:= true;
          exp := premnoeud;premnoeud := premnoeud + 1;
          tabnoeud[exp].tn := vari;
          tabnoeud[exp].reffi := 0;
          tabnoeud[exp].reffr := 0;
          tabnoeud[exp].longch := 1;
          tabnoeud[exp].refch := debut;
          tabnoeud[exp].ordrinit := 1;
          tabnoeud[exp].effectue := false;
          tabnoeud[exp].refsol := 0;
          tabnoeud[exp].methode := 0;
          tabnoeud[exp].etape := 0;
          tabnoeud[exp].refcan := false;
        end
    end; (* Variable *)


  procedure PUISSANCE (exp1,exp2 : refnoeud);

    begin (* Puissance *)
      fin := ff;
      if not place then
        begin
          facteur := false; exp := 0
        end
      else
        begin
          facteur := true;
          exp := premnoeud;premnoeud := premnoeud + 1;
          tabnoeud[exp].tn := pu;
          tabnoeud[exp].reffi := exp1;
```

```
          tabnoeud[exp].reffr := 0;
          tabnoeud[exp].longch := (fin - debut + 1);
          tabnoeud[exp].refch := debut;
          tabnoeud[exp].ordrinit := 1;
          tabnoeud[exp].effectue := false;
          tabnoeud[exp].refsol := 0;
          tabnoeud[exp].methode := 0;
          tabnoeud[exp].etape := 0;
          tabnoeud[exp].refcan := false;
          tabnoeud[exp1].reffr := exp2;
          tabnoeud[exp2].reffr := -exp;
        end
      end; (* Puissance *)


  procedure TRTEXP;


    begin (* Trtexp *)
      if not expression (debut +1 ,f,exp1) then
        begin
          facteur := false;
          exp := 0
        end
      else
        if tabcar [f + 2] <> '^' then
          begin
            facteur := true; exp := exp1; fin := f +1;
            tabnoeud [exp].refch := debut;
            tabnoeud [exp].longch := (fin - debut + 1)
          end
        else (* <EXP> = (<EXP1>) ^ <EXP2> *)

          if not valeur (f +3,ff,exp2) then
            begin
              facteur := false;
              exp := 0
            end
          else
            begin
              puissance (exp1,exp2);
              tabnoeud [exp1].refch := debut;
              tabnoeud [exp1].longch := (f+1-debut+1)
            end
    end; (* Trtexp *)

  procedure TRTVAR;


    begin (* Trtvar *)
      if not variable (debut,f,exp1) then
        begin
          facteur := false;
          exp := 0
        end
      else
        if tabcar [f+1] <> '^' then
          begin
            facteur := true ; exp := exp1; fin := f
          end
        else
          if not valeur (f + 2,ff,exp2) then
```

```
            begin
              facteur := false;
              exp := 0
            end
          else puissance (exp1,exp2)end;(* Trtvar *)


procedure TRTRAT;


  begin (* Trtrat *)
   if not valeur (debut,f,exp1) then
     begin
       facteur := false; exp := 0
     end
   else
     if tabcar [f + 1] <> '^' then
       begin
         facteur := true ; exp := exp1; fin := f
       end
     else
       if not valeur (f+2,ff,exp2) then
         begin
           facteur := false;
           exp := 0
         end
       else puissance (exp1,exp2)
  end; (* Trtrat *)



begin (* Facteur *)
  if tabcar [debut] = '(' then trtexp
  else
    if tabcar [debut] in ['0'..'9'] then trtrat
    else trtvar
end; (* Facteur *)


begin (* Terme *)
    if not facteur (debut,f,exp1) then
      begin
        terme := false;
        exp := 0
      end
    else
      if tabcar [f+1] <> '*' then
        begin
          fin := f; exp := exp1; terme := true
        end
      else
        begin
          termine := false;
          derf := exp1;
          dc := f + 2;
          while not termine do
            begin
              if not facteur (dc,fc,fcrt) then
                begin
                  terme := false ; exp := 0; termine := true
                end
              else
```

```
                    begin
                      tabnoeud[fcrt].ordrinit := tabnoeud[derf].ordrinit
                                                 + 1;
                      tabnoeud[derf].reffr := fcrt;
                      derf := fcrt;
                      if (tabcar [fc + 1] <> '*') then
                        begin
                          fin := fc ; terme := true; termine := true;
                          creerprod
                        end
                      else dc := fc + 2
                    end
                end
            end
    end; (* Terme *)


procedure TRTSOM (premt : refnoeud);

    var dert,tcrt,tcrti : refnoeud;
        dc,fc : integer;
        termine : boolean;

  procedure CREERSOM ;

    begin (* Creersom *)
      if not place then
        begin
          expression := false; exp := 0
        end
      else
        begin
          exp := premnoeud; premnoeud := premnoeud + 1;
          fin := fc;
          tabnoeud[exp].tn := pl;
          tabnoeud[exp].reffi := premt;
          tabnoeud[exp].reffr := 0;
          tabnoeud[exp].longch := (fin - debut + 1);
          tabnoeud[exp].refch := debut;
          tabnoeud[exp].ordrinit := 1;
          tabnoeud[exp].effectue := false;
          tabnoeud[exp].refsol := 0;
          tabnoeud[exp].methode := 0;
          tabnoeud[exp].etape := 0;
          tabnoeud[exp].refcan := false;

          tabnoeud [dert].reffr := -exp
        end
    end ;(* Creersom *)


    begin (* Trtsom *)

      dert := premt; termine := false;
      dc := f + 2;

      while not termine do
        begin
```

```
        case tabcar [dc - 1] of
          '+' : begin
                  if (not terme (dc,fc,tcrt)) then
                    begin
                      expression := false; exp := 0;termine := true
                    end
                  else
                    begin
                      tabnoeud [tcrt].ordrinit :=
                              tabnoeud [dert].ordrinit  + 1;
                      tabnoeud [dert].reffr := tcrt;
                      dert := tcrt;
                      if (not (tabcar [fc+1] in ['+','-'])) then
                        begin
                          fin := fc; expression := true; termine := true;
                          creersom
                        end
                      else dc := fc + 2
                    end
                end;
          '-' : begin
                  if (not terme (dc,fc,tcrti)) then
                    begin
                      expression := false; exp := 0;termine := true
                    end
                  else
                    if not moinsmona (tcrti,tcrt) then
                      begin
                        expression := false; exp := 0; termine := true
                      end
                    else
                      begin
                        tabnoeud [tcrt].ordrinit :=
                                tabnoeud [dert].ordrinit  + 1;
                        tabnoeud [dert].reffr := tcrt;
                        dert := tcrt;
                        if (not (tabcar [fc+1] in ['+','-'])) then
                          begin
                            fin := fc; expression := true;
                            termine := true;
                            creersom
                          end
                        else dc := fc + 2
                      end
                end
        end
    end
end; (* Trtsom *)



begin (* Expression *)
  (* S'occuper d'abord du 1er terme *)

  if tabcar [debut] = '-' then
   begin
     if not terme (debut+1,f,exp1) then
       begin
         expression := false;
         exp := 0
```

```
          end
        else
          if not moinsmona (exp1,expi) then
            begin
              expression := false; exp := 0
            end
          else
            if (not (tabcar [f + 1] in ['+','-'])) then (* On a pas de
                                                          somme *)
              begin (* L'expression decouverte est un moins monadique*)
                fin := f; expression := true; exp := expi
              end
            else (* On a une somme dont expi est le premoer terme *)
              trtsom (expi)
      end
    else
      begin
        if not terme (debut,f,exp1) then
          begin
            expression := false;
            exp := 0
          end
        else
          if (not (tabcar [f+1] in ['+','-'])) then
            begin
              fin := f;expression := true; exp := exp1
            end
          else (* On a une somme dont exp1 est le premoer terme *)
              trtsom (exp1)

      end
  end; (* Expression *)

  begin (* Filsfrere *)
    k := 1;
    while chexp [k] <> chr(32) do k := k + 1;
    k := k - 1;
    if (premcar + k -1) > dercar then
      begin
        filsfrere := false; exp := 0
      end
    else
      begin
        j := premcar;
        for i := 1 to k do
          begin
            tabcar [premcar] := chexp [i];
            premcar := premcar + 1
          end;

        filsfrere := expression(j,f,exp)

      end
  end; (* FILSFRERE *)

begin

end.

(*-----------------------------------*)
```

```
(**S++*)

program TESTFILSFRERE;

uses (**U #5:GLOBAL.CODE*) global,
     (**U #5:FILSFRERE.CODE*) sunbrother,
     (**U #5:LIRE.CODE*) lire;

var ch : char;
    ligne : integer;
    veux : boolean;

procedure PARCOURS (exp : integer);


  var expi,exp1 : integer;
      fin : boolean;


  procedure ECRIRE (exp : integer);


    var n : noeud;
        a,i :integer;

    procedure RESTE;


      begin
        writeln (' ORDRINIT ',n.ordrinit );
        write (' EFFECTUE ');
          if n.effectue then writeln (' TRUE ')
          else writeln (' FALSE ');
        writeln (' REF SOLUTION ',n.refsol );
        writeln (' METHODE ',n.methode );
        writeln (' ETAPE ',n.etape );
        write (' REFCAN ');
          if n.refcan then writeln (' TRUE ')
          else writeln (' FALSE ');
      end;

    begin
      n := tabnoeud [exp];
      write ('TYPENOEUD : ');
        case n.tn of
          pl : writeln ('PL ');
          mu : writeln ('MU ');
          pu : writeln ('PU ');
          mo : writeln ('MO ');
          val: writeln ('VAL ');
          vari : writeln ('VARI ')
        end;
      writeln ('REFFILS : ',n.reffi);
      writeln ('REFFRERE : ',n.reffr);
      writeln ('LONGUEURCHAINE : ',n.longch);
      writeln ('REF CHAINE : ',n.refch);
      writeln ('CHAINE : ');
      writeln (' ');
      a := n.refch;
      for i := 0 to (n.longch - 1) do
        write (tabcar [a + i]);
      writeln (' ');
```

```
    reste
  end;


procedure STACK;

  var ffin : boolean;
      ch : char;


procedure DIR;

  begin (* Dir *)
  end; (* Dir *)


procedure SEQ;


  var fin : boolean;


  begin (* Seq *)
    expi := 1;
    fin := false;

    repeat

      writeln (' COORDONNEE : ',expi);
      ecrire (expi);
      writeln (' TAPE F(in, RET(suite ');
      repeat read (keyboard,ch) until ch in [chr(32),'F'];
      if ch = 'F' then fin := true
      else
        begin
          expi := expi + 1;
          if expi >= premnoeud then fin := true
        end

    until fin

  end; (* Seq *)


  begin (* Stacks *)
    ffin := false;
    repeat

      page(output); gotoxy (0,0);
      writeln (' STACK PROMENADE : ');
      writeln ('   1.SEQUENCIELLE : S ');
      writeln ('   2.DIRECTE : D ');
      writeln ('   3.FIN : F ');
      repeat read (keyboard,ch) until ch in ['S','D','F'];
      case ch of
        'S': seq;
        'D': dir;
        'F': ffin := true
      end;
```

```
    until ffin
  end; (* Stack *)


function CHOIX : char;


  begin
    writeln ('VOTRE CHOIX ?  ');
    writeln ('  I : FILS  ');
    writeln ('  R : FRERE  ');
    writeln ('  F : FIN  ');
    writeln ('  E : ETAT DES STACKS  ');
    repeat read (keyboard,ch) until ch in ['I','R','F','E'];
    choix := ch
  end;



begin (* Parcours *)
  fin := false;

  if exp in [1..nn] then
    begin
      expi := exp;
      ecrire (expi);
      wait
    end
  else expi := 1;

  repeat
    page (output); gotoxy (0,0);

    case choix of

      'I' : begin
              exp1 := tabnoeud[expi].reffi;
              if exp1 > 0 then
                begin
                  expi := exp1;
                  ecrire (expi);
                  wait
                end
              else if exp1 = 0 then
                     begin
                       writeln (' PAS FOU, NON !! EXP = 0  ');
                       wait
                     end
                   else
                     begin
                       expi := -exp1;
                       writeln (' ON REMONTE AU PERE  ');
                       ecrire (expi);
                       wait
                     end
            end;

      'R' : begin
              exp1 := tabnoeud[expi].reffr;
              if exp1 > 0 then
                begin
                  expi := exp1;
```

```
                ecrire (expi);
                wait
              end
            else if exp1 = 0 then
                   begin
                     writeln (' PAS FOU, NON !! EXP = 0  ');
                     wait
                   end
                 else
                   begin
                     expi := -exp1;
                     ecrire (expi);
                     wait
                   end
          end;

     'F' : fin := true;

     'E' : begin
             stack;
             if exp in [1..nn] then expi := exp
             else expi := 1;
             wait
           end
   end

  until fin
end; (* Parcours *)

egin (*Tstfilsfrere*)
 premcar :=1; premnoeud := 1;
 dercar := nc; dernoeud := nn;
 ligne :=9;

 repeat
   repeat

     page (output); gotoxy (0,0);
     writeln (' TAPE YOUR EXPRESSION  ');

   until lirexp (ligne,chexp);

   gotoxy (0,10); writeln ('  NOUS ALLONS CONSTRUIRE L ARBRE FILSFRERE  ');
   wait; page (output);

   if filsfrere (chexp,exp) then
       writeln ('  NOUS AVONS EU ASSEZ DE PLACE ')
   else writeln ('  NOUS N AVONS PAS EU ASSEZ DE PLACE  ');
   writeln ('  EXP  :  ',exp);
   wait; page(output);

   parcours (exp);

   wait; page (output);

   writeln ('  ENCORE CONSTRUIRE UN ARBRE ?  <O,N>  ');
   repeat read (keyboard,ch) until ch in ['O','N'];

   veux := (ch = 'O')

until not veux
```

end. (* Testfilsfrere *)

```
(*=====================================
```

Annexe 4.4.

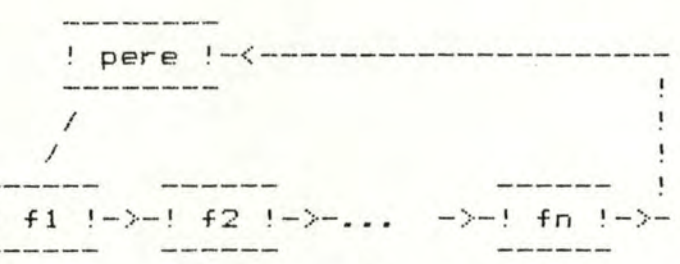
UNIT ORDONNER

```
=======================================
```

4.4.1. SPECIFICATIONS

DESCRIPTION DE L'ORDRE SUR UN ARBRE FILS-FRERE

Pour tout noeud pere de type somme (pl) ou de type produit (mu)

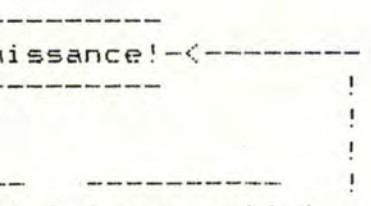


les fils de pere sont ordonnes ssi pour tout 1<=i<=n-1, ordre (Fi) <= ordre (Fi+1) . ORDRE est definie comme suit :

1. ordre sur le type de noeud Tn :
   ordre (Tn) = ORD (Tn)
   rappel: Tn est de type TYPENOEUD enumere
   (pl,mu,pu,val,vari,mo)
   sur lequel il ya un ordre defini .
2. ordre dans un meme type de noeud :
   2.1 si Tb = vari alors ordre alphabetique
   ordre (a) = ordre (b) +1
   2.2 si Tn = val alors ordre numerique

Dans un arbre ordonne, l'ordre dans lequel les operandes apparaissent dans le champs 'chaine de caracteres' de chaque noeud pere est celui de l'ordre de ses enfants (Fi).

L'ordre des deux enfants (base et exposant) d'une puissance est fixe deja a la construction de l'arbre filsfrere .



Les cas de noeud moins, valeur, variable n'ont pas plus qu'un enfants .

FUNCTION ORDARB (EXP)

PRCD : Exp est la coordonnee dans tabnoeud de la racine d'un arbre fils-frere

PSCD : Renvoie vrai si il y a suffisamment de place pour reorganiser l'arbre <exp>
renvoie faux sinon.

COMMENT :

pour un pere, ordonner chaque sous-arbre fils (si ils existent)
ordonner les noeuds racines des sous-arbres fils ENTRE EUX (uniquement s'il y a plusieurs enfants)
mise a jour du champs 'chaine de caractere' du noeud pere

faire la sequence ci-dessus pour chaque noeud pere de l'arbre <exp> .
Ceci suppose donc un parcours postfixe d'arbre fils-frere .

REM :
Les enfants d'un pere sont :
- son fils
- les freres du fils

IMBRIQUE :
ORDSARB .

FUNCTION ORDSARB (PERE : REFNOEUD): BOOLEAN;

PRCD : Pere est la coordonnee dans tabnoeud de la racine d'un arbre fils-frere

PSCD : renvoie vrai si il y a eu assez de place pour ordonner l'arbre <pere>.

COMMENT :
v. ORDONNER

IMBRIQUES :
ORDARBF, ORDRACF, REORGPL, REORGMU, REORGPU, REORGMO, RECOPIER, SIGNE, PARPERE.

PROCEDURE ORDARBF (PERE : REFNOEUD);

PRCD : Pere est la coordonnee de la racine d'une expression

PSCD : renvoie orsarb a vrai si il ya assez de place pour ordonner tous les fils de <pere>
renvoie faux sinon.

PROCEDURE ORDRACF;

PRCD: Pere est la coordonnee de la racine d'un arbre dont chaque arbre fils est ordonne.

PSCD: La chaine des fils est ordonnee suivant l' ordre defini ci-dessus.

COMMENT:
L'ordre de chaque fils est comparer à l'ordre du fils suivant. Si l'ordre du dernier fils envisage est strictement inferieur a l'avant dernier, alors permuter dans la chaine des fils.

PROCEDURE REORGPL;

PRCD: <pere> est une somme dont les fils sont ordonnes et ordonnes entre eux.

PSCD: renvoie ordsarb a vrai si il ya eu assez de place pour adapter le champs "chaine de caracteres" de <pere> aux permutations de ses fils.

COMMENT:
1.prendre chaque fils dans leur nouvel ordre et construire la chaine qui represente leur somme dans une "zone auxiliaire droite" (ZAD). La ZAD est une partie de tabcar reservee.

2.recopier la ZAD "a gauche" dans la zone de tabcar reservee a pere.

S'il n'y a pas assez de place pour ces operations de recopiage, ordsarb vaudra false.
Rem: il faut detecte la perte de longueur significative et ajouter un ' ' si necessaire. Il y a perte quand le premier caractere de la chaine initiale est un '-' et que le premier caractere significatif de la chaine apres reorganisation est <> '-'.

IMBRIQUES:
MAJPERE.

PROCEDURE REORGMU;

PRCD: <pere> est une produit dont les fils sont ordonnes et ordonnes entre eux.

PSCD: renvoie ordsarb a vrai si il ya eu assez de place pour adapter le champs "chaine de caracteres" de <pere> aux permutations de ses fils.

COMMENT:
1.prendre chaque fils dans leur nouvel ordre et construire la chaine qui represente leur produit dans une "zone auxiliaire droite" (ZAD). La ZAD est une partie de tabcar reservee.

2.recopier la ZAD "a gauche" dans la zone de tabcar reservee a pere.

S'il n'y a pas assez de place pour ces operations de recopiage, ordsarb vaudra false.

IMBRIQUES:
MAJPERE.

PROCEDURE REORGMO;

PRCD: <pere> est une expression moins.

PSCD: renvoie ordsarb a vrai si il ya eu assez de place pour ordonner le fils de <pere> et pour pour adapter le champs "chaine de caracteres" de <pere> aux modifications que son fils a subi.

COMMENT: recopier la chaine representant son fils ordonne precedee du signe "-".

PROCEDURE REORGPU;

PRCD: <pere> est une puissance.

PSCD: renvoie ordsarb a vrai si il ya eu assez de place pour ordonner la base de la puissance (fils de <pere>) et pour adapter le champs "chaine de caracteres" de <pere> aux modifications que son fils a subi.

PROCEDURE SIGNE (VAR R : INTEGER; C: CHAR);

PRCD: R est une coordonnee dans tabcar. C est un caractere.

PSCD: Tabcar[r] = 'c' et r est augmente de 1.

PROCEDURE RECOPIER (VAR R1,R2: INTEGER; L : INTEGER);

PRCD: R1, r2 sont des coordonnees dans tabcar. L est un entier representant le nombre de caracteres a recopier.

PSCD: Tous les caracteres de la zone r1..r1+l-1 sont recopies dans la zone r2..r2+l-1. R1 et r2 sont incrementes de l.

FUNCTION PARPERE (PERE: REFNOEUD): INTEGER;

PRCD: Pere est la coordonnee dans tabnoeud de la racine d'une expression.

PSCD: Renvoie le nombre de parenthese propres de la racine de <exp>.

```
DEFINITION de parentheses "propres":
  les parentheses propres d'un noeud sont des parentheses qui
  entourent ce noeud et qui ne se retrouvent pas chez son fils
  (s'il existe).
Exemple:
  noeud:((a)) parpere:2
  noeud:(a*b)+(a*c) parpere:0 car la 1ere parenthese se retrouve
        chez le fils (a*c).
-----------------------------------*)
```

```
(*
4.4.2.CODE
      ----*)


(*$S+,N+*)
unit ORDONNER;

interface

  uses (*$U #5:GLOBAL.CODE*) global;
  function ORDARB (exp : refnoeud) : boolean;

implementation


function ORDARB;


function ORDSARB (pere : refnoeud) : boolean;

  var ordcrt : boolean;
      frere,fils : refnoeud;


  procedure ORDARBF; (* Ordonner chaque sous-arbre fils *)


    var enfcrt : refnoeud;


    begin (* Ordarbf *)
      enfcrt := tabnoeud [pere].reffi;
      while enfcrt > 0 do
        begin
          ordsarb := ordsarb (enfcrt);
          enfcrt := tabnoeud [enfcrt].reffr
        end;
    end; (* Ordarbf *)


  procedure ORDRACF;


    var prem,der,nouv : refnoeud;


      function ORDINF (n1,n2 : refnoeud ) : boolean;

        type comp = (inf,sup,eq);

        var car : char;
```

```
function COMPORD (n1,n2 : refnoeud) : comp;

  var t1,t2 : typenoeud;

  procedure MUPL;

    var crt1,crt2 : refnoeud;
        fin : boolean;
    begin
      crt1 := tabnoeud [n1].reffi;
      crt2 := tabnoeud [n2].reffi;
      fin := false;
      repeat
        case compord (crt1,crt2) of
          inf : begin compord := inf; fin := true end;
          sup : begin compord := sup; fin := true end;
          eq  :
            begin
              crt1 := tabnoeud [crt1].reffr;
              crt2 := tabnoeud [crt2].reffr;
              if ((crt1 > 0) or (crt2 > 0)) then
                if ((crt1 <= 0) and (crt2 <= 0)) then
                  begin
                    fin := true; compord := eq
                  end
                else
                  if (crt1 <= 0) then
                    begin
                      fin := true; compord := inf
                    end
                  else
                    begin
                      fin := true; compord := sup
                    end
            end
        end
      until fin
    end;

function ORDNUM (n1,n2 : refnoeud) : comp;

  var l1,l2,ref1,ref2,i : integer;
      fin : boolean;

  begin (* Ordnum *)
    l1 := tabnoeud [n1].longch;
    l2 := tabnoeud [n2].longch;
    if l1 <> l2 then
      if l1 < l2 then ordnum := inf
      else ordnum := sup
    else
      begin
        ref1 := tabnoeud [n1].refch;
        ref2 := tabnoeud [n2].refch;
        fin := false; i := 0;

        repeat
          if (ord(tabcar [ref1 + i]) < ord(tabcar [ref2 + i])) then
            begin
              fin := true; ordnum := inf
            end
```

```
            else
              if (ord(tabcar [ref1 + i]) > ord(tabcar [ref2 + i]))
               then
                begin
                  fin := true; ordnum := sup
                end
              else
                begin
                  i := i + 1;
                  if i > (l1 - 1) then
                    begin
                      fin := true; ordnum := eq
                    end
                end
          until fin
        end
    end; (* Ordnum *)


  function ORDALPHA (n1,n2 : refnoeud) : comp;

    var ch1,ch2 : char;

    begin (* Ordalpha *)
      ch1 := tabcar [tabnoeud [n1].refch];
      ch2 := tabcar [tabnoeud [n2].refch];
      if (ord(ch1) < ord (ch2)) then ordalpha := inf
      else
        if (ord(ch1) > ord (ch2)) then ordalpha := sup
        else
        ordalpha := eq
    end; (* Ordalpha *)



    begin (* Compord *)
      t1 := tabnoeud [n1].tn;
      t2 := tabnoeud [n2].tn;
      if t1 <> t2 then
        if (ord(t1) < ord (t2)) then compord := inf
        else compord := sup
      else
        case t1 of
          val : compord := ordnum (n1,n2);
          vari: compord := ordalpha (n1,n2);
          mo  : compord :=
                        compord (tabnoeud [n1].reffi,
                                 tabnoeud [n2].reffi);
          pu  :
            case compord (tabnoeud[n1].reffi,tabnoeud[n2].reffi) of
              inf : compord := inf;
              sup : compord := sup;
              eq  :
                compord
                := compord (tabnoeud [tabnoeud[n1].reffi].reffr,
                            tabnoeud [tabnoeud[n2].reffi].reffr)
            end;

          mu,pl : mupl
        end
    end; (* Compord *)
```

```
    begin (* Ordinf *)
      case compord (n1,n2) of
        inf : ordinf := true;
        sup : ordinf := false;
        eq  : ordinf := true
      end;
    end; (* Ordinf *)

procedure INSERER;

  var crt :refnoeud;
      fin : boolean;

  begin (* Inserer *)
    if ordinf (nouv,prem) then
      begin
        tabnoeud[der].reffr := tabnoeud[nouv].reffr;
        tabnoeud[nouv].reffr := prem;
        prem := nouv
      end
    else
      if not ordinf (nouv,der) then der := nouv
      else
        begin
          tabnoeud[der].reffr := tabnoeud[nouv].reffr;
          crt := prem;fin := false;
          while not fin do
            begin
              if ordinf (nouv,tabnoeud[crt].reffr) then
                begin
                  fin := true;
                  tabnoeud[nouv].reffr := tabnoeud[crt].reffr;
                  tabnoeud[crt].reffr := nouv
                end
              else crt := tabnoeud[crt].reffr
            end
        end
  end; (* INSERER *)




begin (* Ordracf *)
  prem := tabnoeud [pere].reffi;
  der := prem;
  nouv := tabnoeud [der].reffr;

  while nouv > 0 do
    begin
      inserer;
      nouv := tabnoeud [der].reffr;
    end;
  tabnoeud [pere].reffi := prem;
  tabnoeud [der].reffr := -pere;
end; (* Ordracf *)
```

```
  procedure RECOPIER (var r1,r2 : integer;l : integer);
    var i : integer;
    begin
      for i := 1 to l do
        begin
          tabcar[r2] := tabcar[r1];
          r2 := r2 + 1;r1 := r1 + 1
        end
    end;


  procedure SIGNE (var r : integer;c : char);
    begin
      tabcar[r] := c; r := r + 1
    end;


function PARPERE (pere : refnoeud) : integer;

  var np1,np2,r1,r2 : integer;

  begin (* Parpere *)
    r1 := tabnoeud[pere].refch;
    np1 := 0;
    while (tabcar [r1] in ['(',' '] do
      begin
        if (tabcar[r1] = '(') then np1:= np1 + 1;
        r1 := r1 + 1
      end;

    if (tabnoeud[pere].tn in [vari,val]) then parpere := np1
    else
      begin
        r2 := tabnoeud[tabnoeud[pere].reffi].refch;
        np2 := 0;
        while (tabcar [r2] in ['(',' '] do
          begin
            if (tabcar[r2] = '(') then np2 := np2 +1;
            r2 := r2 + 1
          end;
        if (np1-np2) > 0 then parpere := (np1 - np2)
        else parpere := 0
      end
  end; (* Parpere *)


procedure REORGMU;

  var longchpere,refchpere,np : integer;


procedure MAJPERE;

  var nbrecar : integer;

  procedure RECOPIER;

    var enfcrt,r1,r2,l,i : integer;

    begin (* Recopier *)
      enfcrt := tabnoeud[pere].reffi;
      r2 := dercar + 1;
```

```
      repeat
        l := tabnoeud [enfcrt].longch ;
        r1 := tabnoeud [enfcrt].refch;
        nbrecar := nbrecar + l;
        for i := 1 to l do
          begin
            tabcar[r2]:=tabcar[r1];
            r2 := r2 + 1;r1 := r1 + 1;
          end;
        enfcrt := tabnoeud[enfcrt].reffr;
        if enfcrt > 0 then
          begin
            tabcar[r2] := '*';r2 := r2 + 1;
            nbrecar := nbrecar + 1;
          end

      until enfcrt < 0;
  end; (* Recopier *)


procedure RAMENER;

  var r1,r2,i,refcrt : integer;

  begin (* Ramener *)
    tabnoeud [pere].refch := premcar;
    r1 := premcar; r2 := dercar +1;

    (* Reecrire la ZAD sur la zone initiale apres le '' et les ( propres *)
    if np > 0 then
      for i := 1 to np do
        begin
          tabcar[r1] := '('; r1 := r1 + 1
        end;

    for i := 1 to nbrecar do
      begin
        tabcar [r1] := tabcar[r2];
        r1 := r1 + 1;
        r2 := r2 + 1
      end;
    if np > 0 then
      for i := 1 to np do
        begin
          tabcar[r1] := ')'; r1 := r1 + 1;
        end;
  end; (* Ramener *)


  begin (* Majpere *)
    (* Reserver la place por la ZAD *)
    dercar := dercar - longchpere;

    (* Ecrire dans la ZAD et modifier les references des fils *)
    nbrecar := 0;
    recopier;

    (* Ramener dans la zone initiale *)
    ramener;
    dercar := dercar + longchpere;
    premcar := premcar + longchpere;
  end; (* Majpere *)
```

```
  begin (*Reorgmu *)
    ordcrt := true;

    (* Detecter les '(' PROPRES du pere *)
    np := parpere (pere);

    (* Tester s'il y a suffisamment de place pour la ZAD *)
    longchpere := tabnoeud [pere].longch;
    refchpere := tabnoeud [pere].refch;
    if (dercar - longchpere) < premcar then ordcrt := false
    else
      begin

        ordarbf; (* Ordonner chaque arbre enfant de pere *)

        ordracf; (* Ordonner les enfants de pere ENTRE EUX *)

        majpere;  (* M-a-j du champs "chaine de carac. du pere *)

      end;
    ordsarb := ordcrt;
  end; (* Reorgmu *)


procedure REORGPL;

  var longchpere,refchpere,np : integer;
      susceptible,dimlong : boolean;


procedure MAJPERE;

  var nbrecar : integer;

  procedure RECOPIER;

    var enfcrt,r1,r2,l,b,k,i : integer;

    begin (* Recopier *)
      r2 := dercar + 1;
      enfcrt := tabnoeud [pere].reffi;
      repeat
        l := tabnoeud [enfcrt].longch;
        r1 := tabnoeud [enfcrt].refch;
        nbrecar := nbrecar + l;
        for i := 1 to l do
          begin
            tabcar[r2]:=tabcar[r1];
            r2 := r2 + 1;r1 := r1 + 1;
          end;
        enfcrt := tabnoeud[enfcrt].reffr;
        if enfcrt > 0 then
          begin
            b := 0; k := tabnoeud[enfcrt].refch;
            while tabcar [k] = ' ' do
              begin
                b := b+ 1;
                k := k + 1
```

```
            end;
          if tabcar [tabnoeud[enfcrt].refch + b] <> '-' then
            begin
              nbrecar := nbrecar + 1;
              tabcar [r2] := '+'; r2 := r2 + 1;
            end
        end
    until enfcrt < 0;
  end; (* Recopier *)


procedure RAMENER;

  var r1,r2,i,refcrt : integer;

  begin (* Ramener *)
    tabnoeud[pere].refch := premcar;
    r1 := premcar; r2 := dercar +1;

    (* Assure l'eventuelle diminution de longueur significative *)
    if dimlong then
      begin
        tabcar [r1] := ' ';
        r1 := r1 + 1
      end;

    (* Reecrire la ZAD sur la zone initiale apres le '' et les ( propres
       ---------------------------------
      .!''! (((... !     ZAD    ! ...))))!
       ---------------------------------
       ^
       refchpere                        NP
       !              longchpere          !
    *)
    if np > 0 then
      for i := 1 to np do
        begin
          tabcar[r1] := '('; r1 := r1 + 1;
        end;

    for i := 1 to nbrecar do
      begin
        tabcar [r1] := tabcar[r2];
        r1 := r1 + 1;
        r2 := r2 + 1
      end;
    if np > 0 then
      for i := 1 to np do
        begin
          tabcar[r1] := ')'; r1 := r1 + 1;
        end;
  end; (* Ramener *)


  begin (* Majpere *)

    (* Reserver la place por la ZAD *)
    dercar := dercar - longchpere;

    (* Tester la diminution de longueur significative *)
    dimlong := (susceptible and
                (tabcar[tabnoeud[tabnoeud[pere].reffi].refch] <> '-'));
```

```
    (* Ecrire dans la ZAD et modifier les references des fils *)
    nbrecar := 0;
    recopier;

    (* Ramener dans la zone initiale *)
    ramener;
    dercar := dercar + longchpere;
    premcar := premcar + longchpere;
  end; (* Majpere *)


  begin (*Reorgpl *)

    ordcrt := true;

    (* Detecter les '(' PROPRES du pere *)
    np := parpere (pere);

    (* Tester s'il y a suffisamment de place pour la ZAD *)
    longchpere := tabnoeud [pere].longch;
    refchpere := tabnoeud [pere].refch;
    if (dercar - longchpere) < premcar then ordcrt := false
    else
      begin
      (* Detecter une eventuelle situation de diminution de longueur
              significative *)
        susceptible := (tabcar [tabnoeud[tabnoeud[pere].reffi].refch]
                        = '-');

        ordarbf; (* Ordonner chaque arbre enfant de pere *)

        ordracf; (* Ordonner les enfants de pere ENTRE EUX *)

        majpere;  (* M-a-j du champs "chaine de carac. du pere *)

      end;
  end; (* Reorgpu *)


reorgpu (pere,fils,frere : refnoeud) : boolean;

  var r1,r2,l,np,i : integer;

  begin (* REORGPU *)
    if ( not ordsarb(fils)) then reorgpu := false
    else
      begin
        l := np + np + 1 + tabnoeud[frere].longch + tabnoeud[fils].longch;
        if ((premcar + l) > dercar) then reorgpu := false
        else
          begin
            reorgpu := true;
            r1 := tabnoeud[fils].refch;
            r2 := premcar;
            tabnoeud[pere].refch := premcar;
            if np>0 then for i:= 1 to np do signe (r2,'(');
            recopier (r1,r2,tabnoeud[fils].longch);
            if np>0 then for i:= 1 to np do signe (r2,')');
            signe (r2,'^');
            r1 := tabnoeud[frere].refch;
            recopier (r1,r2,tabnoeud[frere].longch);
```

```
            tabnoeud[pere].longch := l;
            premcar := premcar + l
          end
      end;
  end; (* REORGPU *)

function  REORGMO (pere,fils : refnoeud) : boolean;

  var r1,r2,l,np,i : integer;

  begin (* REORGMO *)
    if ( not ordsarb(fils)) then reorgmo := false
    else
      begin
        np := parpere (pere);
        l := np + np + 1 + tabnoeud[fils].longch;
        if ((premcar + l) > dercar) then reorgmo := false
        else
          begin
            reorgmo := true;
            r1 := tabnoeud[fils].refch;
            r2 := premcar;
            tabnoeud[pere].refch := premcar;
            signe (r2,'-');
            if np>0 then for i:= 1 to np do signe (r2,'(');
            recopier (r1,r2,tabnoeud[fils].longch);
            if np>0 then for i:= 1 to np do signe (r2,')');
            tabnoeud[pere].longch := l;
            premcar := premcar + l
          end
      end;
  end; (* REORGMO *)


  begin (* Ordsarb *)
    if pere > 0 then
      case tabnoeud [pere].tn of
        pl : reorgpl;

        mu : reorgmu;

        pu : begin
               fils := tabnoeud [pere].reffi;
               frere:= tabnoeud [fils].reffr;
               ordsarb := reorgpu (pere,fils,frere)
             end;

        mo : begin
               fils := tabnoeud [pere].reffi;
               ordsarb := reorgmo (pere,fils)
             end;

        vari,val : ordsarb := true

      end;
  end; (* Ordsarb *)
  begin
    ordarb := ordsarb (exp);
  end;

begin
```

```
end.
-----------------------------------------

4.4.3.test
      ----

  (v. 4.5.3.)

-----------------------------------------*)
```

```
(*=====================================

                Annexe 4.5.

              UNIT UTILITAIRES

=====================================.
```

4.5.1.SPECIFICATIONS

```
FUNCTION EGALUN (EXP: REFNOEUD): BOOLEAN;

  PRCD: Exp est la coordonnee de la racine d' une expression.

  PSCD: Renvoie vrai si <exp> represente l'expression "1".

        ------------------------

FUNCTION EGALZERO (EXP: REFNOEUD): BOOLEAN;

  PRCD: Exp est la coordonnee de la racine d' une expression.

  PSCD: Renvoie vrai si <exp> represente l'expression "0".

        ------------------------

FUNCTION UN (VAR EXPF: REFNOEUD): BOOLEAN;;

  PRCD: Expf est une  variable coordonnee dans tabnoeud.

  PSCD: renvoie vrai si il ya eu assez de place pour construire l'
        expression "1" auquel cas expf est la coordonnee de la racine de
        l'expression "1" construite.

        ------------------------

FUNCTION ZERO (VAR EXPF: REFNOEUD): BOOLEAN;;

  PRCD: Expf est une  variable coordonnee dans tabnoeud.

  PSCD: renvoie vrai si il ya eu assez de place pour construire l'
        expression "0" auquel cas expf est la coordonnee de la racine de
        l'expression "0" construite.

        ------------------------

FUNCTION CARACNBRE (E: REFNOEUD): INTEGER;

  PRCD: E est la racine d' une expression valeur qui represente un entier.

  PSCD: Renvoie l'entier represente par <e>.
```

```
FUNCTION ENTIERCARAC (ENTIER: INTEGER; VAR REF, LONG: INTEGER): BOOLEAN;
```

PRCD: Entier est un entier, ref et long des references dans tabcar.

PSCD: Renvoie vrai si il y asse z de place pour representer entier dans tabcar auquel cas renvoie la coordonne ref et la longueur long de la representation sous forme de chaine de caracteres de l'entier entier.

```
FUNCTION VALEUR (VAR EXP REFNOEUD; D : INTEGER): BOOLEAN;
```

PRCD: Exp est une reference dans tabnoeud. D est un entier.

PSCD: Renvoie vrai si il y a assez de place pour construire l'arbre <exp> representant l'entier d.
Renvoie faux sinon.

```
FUNCTION CHEGAL (EXP1,EXP2: REFNOEUD): BOOLEAN;;
```

PRCD: <exp1> et <exp2> sont des arbre fils-freres.

PSCD: renvoie vrai si les chaines de caracteres incluses dans les representations sous forme d'arbre de <exp1> et <exp2> sont identiques.

DEFINITION: de l'identite de deux chaines
Deux chaines sont identiques si elles comprennent les memes symboles et dans le meme ordre aux "blancs" et aux parenthese propres pres.
Exemple:
((a*b)) = a*b = (a*b) <> (a)*(b)
a*b+c+(a-b) = a*b+c+(a-b).
Rem: voir definition de parenthese propres dans les spec. du module ordonner.

COMMENT:
1. Rechercher le declage gauche et le decalage droit: references dans tabcar du premier et du dernier caracteres <> parenthese propre et ceci pour <exp1> et <exp2>.
2. A partir des decalages gauches jusqu'aux decalages droits, comparer les chaines de <exp1> et <exp2> aux "blancs pres".

IMBRIQUES:
DECALAGE.

```
FUNCTION BASE (EXP: REFNOEUD): REFNOEUD;;
```

PRCD: Exp est la coordonnee de la racine d'une expression.

PSCD: Renvoie la coordonnee de la racine de la base de <exp> si <exp> ext une puissance, renvoie exp sinon.

```
FUNCTION DUPLICATA (PERE1: REFNOEUD; VAR PERE2: REFNOEUD): BOOLEAN;
```

PRCD: Pere1 est la coordonnee de la racine d'une expression. Pere2 est une variable coordonnee dans tabnoeud.

PSCD: Renvoie vrai si il y assez de place pour creer l'arbre fils-frere <pere2> ayant exactement la meme structure que <pere1>.

IMBRIQUES:
CREERNOEUD.

----------*)

```
(*
4.5.2.CODE
      -----*)


(*$S+*)

unit UTILITAIRES;

interface

 uses (*$U #5:GLOBAL.CODE*) global;

 function EGALUN (exp : refnoeud) : boolean;
 function EGALZERO (exp : refnoeud) : boolean;
 function UN (var expf : refnoeud) : boolean;
 function ZERO (var expf : refnoeud) : boolean;
 function CARACNBRE (e : refnoeud) : integer;
 function ENTIERCARAC (entier : integer;var ref,long : integer) : boolean;
 function VALEUR (var exp :refnoeud; d : integer) : boolean;
 function  CHEGAL (exp1,exp2 : refnoeud): boolean;
 function BASE (exp : refnoeud) : refnoeud;
 function DUPLICATA (pere1 : refnoeud; var pere2 : refnoeud ) : boolean;


implementation


  function EGALUN;

    var nnoeud : noeud;

    begin (* Egalun *)
      nnoeud := tabnoeud [exp];
      egalun := ((tabcar [nnoeud.refch] = '1')
                 and (nnoeud.longch = 1))
    end; (* Egalun *)


  function EGALZERO;

    var nnoeud : noeud;

    begin
      nnoeud := tabnoeud [exp];
      egalzero:= ((tabcar [nnoeud.refch] = '0')
                 and (nnoeud.longch = 1))
    end;

function UN;

    begin
      if (not place or (dercar < (premcar +1))) then un := false
      else
        begin
          un := true;
          expf := premnoeud; premnoeud := premnoeud + 1;
          tabnoeud [expf].tn:=val;
          tabnoeud [expf].reffi:=0;
```

```
          tabnoeud [expf].reffr:=0;
          tabnoeud [expf].longch:=1;
          tabnoeud [expf].refch:=premcar;premcar := premcar + 1;;
          tabnoeud [expf].ordrinit:=1;
          tabnoeud [expf].refsol:=0;
          tabnoeud [expf].methode:=0;
          tabnoeud [expf].etape:=0;
          tabnoeud [expf].refcan:=false;
          tabcar [tabnoeud[expf].refch]:='1'
        end
    end;

  function ZERO;

    begin
      if (not place or (dercar < (premcar +1))) then zero := false
      else
        begin
          zero := true;
          expf := premnoeud; premnoeud := premnoeud + 1;
          tabnoeud [expf].tn:=val;
          tabnoeud [expf].reffi:=0;
          tabnoeud [expf].reffr:=0;
          tabnoeud [expf].longch:=1;
          tabnoeud [expf].refch:=premcar;premcar := premcar + 1;;
          tabnoeud [expf].ordrinit:=1;
          tabnoeud [expf].refsol:=0;
          tabnoeud [expf].methode:=0;
          tabnoeud [expf].etape:=0;
          tabnoeud [expf].refcan:=false;
          tabcar [tabnoeud[expf].refch]:='0'
        end
    end;

  function CARACNBRE;

    var r1,i,a : integer;

    function TRANSFO (ref,long : integer) : integer;

      var i,r,l,n,t,j,k : integer;

      begin (* Transfo *)
        i := 1; r := ref; l := long - 1; t := 0;
        while i <= long do
          begin
            case tabcar [r] of
              '0': n := 0;
              '1': n := 1;
              '2': n := 2;
              '3': n := 3;
              '4': n := 4;
              '5': n := 5;
              '6': n := 6;
              '7': n := 7;
              '8': n := 8;
              '9': n := 9
            end;
            if (l = 0) then j := 1
            else
              begin
                j := 1;
```

```
            for k := 1 to l do
              j := j*10;
          end;
        t := (n*j)+t;
        r := r + 1; l := l - 1;i := i + 1
      end;
    transfo := t
  end; (* Transfo *)

begin (* Caracnbre *)

  a := transfo (tabnoeud [e].refch,tabnoeud[e].longch);
  caracnbre := a;

end; (* Caracnbre *)


function ENTIERCARAC;

  var divid,quot,reste,i,r1 : integer;
      tjrsplace : boolean;

  begin (* Entiercarac *)
    divid := entier; long := 0;
    tjrsplace := true;

    repeat
      long := long + 1;
      quot := divid div 10;
      reste := divid mod 10;
      if ((dercar - 1) < premcar) then tjrsplace := false
      else
        begin
          case reste of
            0 : tabcar [dercar] := '0';
            1 : tabcar [dercar] := '1';
            2 : tabcar [dercar] := '2';
            3 : tabcar [dercar] := '3';
            4 : tabcar [dercar] := '4';
            5 : tabcar [dercar] := '5';
            6 : tabcar [dercar] := '6';
            7 : tabcar [dercar] := '7';
            8 : tabcar [dercar] := '8';
            9 : tabcar [dercar] := '9'
          end;
          dercar := dercar - 1;
          divid := quot
        end

    until ((divid = 0) or (not tjrsplace));

    if (not tjrsplace) then entiercarac := false
    else
      begin
        ref := premcar; entiercarac := true;
        i := 1;
        while i <= long do
          begin
            dercar := dercar + 1;
            tabcar [premcar] := tabcar [dercar];
            premcar := premcar + 1;
```

```
            i := i + 1
          end
      end;
  end; (* Enrtiercarac *)


  function VALEUR;

    var r,l : integer;

    begin (* Valeur *)
      if ((not place) or (not entiercarac (d,r,l))) then valeur := false
      else
        begin
          valeur := true; exp := premnoeud; premnoeud := premnoeud + 1;
          tabnoeud [exp].tn := val;
          tabnoeud [exp].reffi:=0;
          tabnoeud [exp].reffr:=0;
          tabnoeud [exp].longch:=1;
          tabnoeud [exp].refch:= r;
          tabnoeud [exp].ordrinit:=1;
          tabnoeud [exp].refsol:=0;
          tabnoeud [exp].methode:=0;
          tabnoeud [exp].etape:=0;
          tabnoeud [exp].refcan:=false;
        end;
    end; (* Valeur *)


function  CHEGAL;

  var r1,r2,f1,f2 : integer;
      egal,fin1,fin2 : boolean;
      t1,t2 : typenoeud;

  procedure DECALAGE (exp : refnoeud; var decg,decd : integer);

    var np1,np2,np,r1,r2,r,d : integer;
        fin : boolean;

    begin (* Decalage *)
      np1 := 0; np2 := 0;
      r1 := tabnoeud[exp].refch;

      r2 := tabnoeud[tabnoeud[exp].reffi].refch;
      (* Nbre de parentheses au debut de pere *)
      fin := false;
      while not fin do
        begin
          if tabcar [r1] = ' ' then r1 := r1 + 1
          else
            if tabcar [r1] = '(' then
              begin
                np1 := np1 + 1;
                r1 := r1 + 1
              end
            else fin := true
        end;
      (* Nbre de parentheses au debut du fils *)
      fin := false;
      while not fin do
        begin
```

```
        if tabcar [r2] = ' ' then r2 := r2 + 1
        else
          if tabcar [r2] = '(' then
            begin
              np2 := np2 + 1;
              r2 := r2 + 1
            end
          else fin := true
      end;
    (* Nbre de parentheses propres au pere *)
    if (np1 - np2) > 0 then np := (np1 - np2)
    else np := 0;
    (* Decalage droit et decalage gauche *)
    decd := tabnoeud[exp].refch+tabnoeud[exp].longch - 1 - np;
    r := tabnoeud[exp].refch ;
    if (np <> 0) then
      begin
        d := 0;
        repeat
          begin
            if tabcar[r] = ' ' then r := r + 1
            else
              if (tabcar[r] = '(') then
                begin
                  r := r + 1; d := d + 1
                end
          end
        until d = np
      end;
    while (tabcar [r] = ' ') do r  := r +1;
    decg := r;
  end; (* Decalage *)

function EGALCH (exp1,exp2 : refnoeud) : boolean;

  var r1,r2,f1,f2 : integer;
      egal : boolean;

  begin
    r1 := tabnoeud[exp1].refch;
    r2 := tabnoeud[exp2].refch;
    f1 := tabnoeud[exp1].longch + r1 -1;
    f2 := tabnoeud[exp2].longch + r2 -1;
    while (tabcar[r1] = '(') do r1 := r1 +1;
    while (tabcar[f1] = ')') do f1 := f1 -1;
    while (tabcar[r2] = '(') do r2 := r2 +1;
    while (tabcar[f2] = ')') do f2 := f2 -1;
    if ((f1 - r1) <> (f2 - r2)) then chegal := false
    else
      begin
        egal := true;
        while ((r1 <= f1) and egal) do
          begin
            egal := (tabcar[r1] = tabcar[r2]) ;
            r1 := r1 + 1; r2 := r2 + 1
          end;
        egalch := egal
      end
  end;

begin (* Chegal *)
  t1 := tabnoeud[exp1].tn;
```

```
    t2 := tabnoeud[exp2].tn;
    if (t1 <> t2) then chegal := egalch (exp1,exp2)
    else
     if (t1 in [vari,val]) then chegal := egalch (exp1,exp2)
     else
        begin
          egal := true;
          decalage (exp1,r1,f1);
          decalage (exp2,r2,f2);
          fin1 := false; fin2 := false;
          while (egal and not (fin1 or fin2)) do
            begin
              while ((tabcar[r1] = ' ') and (not fin1)) do
                begin
                  r1 := r1 + 1; fin1 := (r1 > f1)
                end;
              while ((tabcar[r2] = ' ') and (not fin2)) do
                b)gin
                  r2 := r2 + 1; fin2 := (r2 > f2)
                end;
              if ((not fin1) and (not fin2)) then
                begin
                  egal := (tabcar[r1] = tabcar[r2]);
                  r1 := r1 + 1; r2 := r2 + 1;
                  fin1 :=  (r1 > f1); fin2 := (r2 > f2)
                end
            end;

          if egal then
            if fin1 then
              if fin2 then chegal := true
              else
                begin
                  while ((r2 <= f2)and egal) do
                    begin
                      egal := (tabcar[r2] = ' ');
                      r2 := r2 + 1
                    end;
                  chegal := egal
                end
            else
              if fin2 then
                begin
                  while ((r1 <= f1) and egal) do
                    begin
                      egal := (tabcar[r1] = ' ');
                      r1 := r1 + 1
                    end;
                  chegal := egal
                end
            else chegal := false
        end;
    end; (* Chegal *)

function BASE;

  begin
    if tabnoeud[exp].tn <> pu then base := exp
    else base := tabnoeud [exp].reffi
  end;
```

```
function DUPLICATA;

    var fils2,enfcrt1,der2,enfcrt2 : refnoeud;
        tjrsplace : boolean;

    function CREERNOEUD (exp1 : refnoeud;var exp2 : refnoeud) : boolean;

      begin (* Creernoeud *)
        if not place then creernoeud := false
        else
          begin
            creernoeud := true;
            exp2 := premnoeud; premnoeud := premnoeud + 1;
            tabnoeud[exp2] := tabnoeud[exp1];
            tabnoeud[exp2].reffi := 0;
            tabnoeud[exp2].reffr := 0;
            tabnoeud[exp2].ordrinit := 1;
          end
      end; (* Creernoeud *)


    begin (* Duplicata *)
      duplicata := true;
      (* Creer d'abord le noeud pere *)
      if not creernoeud (pere1,pere2) then duplicata := false
      else
       if (tabnoeud[pere1].reffi = 0) then tabnoeud[pere2].reffi := 0
       else
         (* Ensuite, creer tous les fils s'il y en a*)
         if (not duplicata (tabnoeud[pere1].reffi,fils2)) then
           duplicata := false
         else
           begin
             tabnoeud [pere2].reffi := fils2;
             enfcrt1 := tabnoeud[tabnoeud[pere1].reffi].reffr;
             der2 := fils2;
             tjrsplace := true;

             while ((enfcrt1 > 0) and tjrsplace) do
               begin
                 if not duplicata (enfcrt1,enfcrt2) then tjrsplace :=
alse

                 else
                   begin
                     tabnoeud[der2].reffr := enfcrt2;
                     der2 := enfcrt2; enfcrt1 := tabnoeud[enfcrt1].reffr
                   end
               end;

             if tjrsplace then tabnoeud[der2].reffr := -pere2
             else duplicata := false
           end;
    end; (* Duplicata *)


  begin
  end.

  (*-----------------------------------*)
```

```
(*
4.5.3.TEST
      ----*)


(*$S++*)

program TESTUTILITAIRE;

uses (*$U #5:GLOBAL.CODE*) global,
     (*$U #5:FILSFRERE.CODE*) sunbrother,
     (*$U #5:LIRE.CODE*) lire,
     (*$U #5:ORD.CODE*) ordonner,
     (*$U #5:UT.CODE*) utilitaires;

var ch : char;
    ligne : integer;
    veux : boolean;

procedure TEST1;

  begin
    repeat
      page (output); gotoxy(0,0);
      writeln ('TAPE L EXPRESSION ');
    until lirexp (ligne,chexp);
    if filsfrere (chexp,exp) then
      begin
        writeln ('ASSEZ DE PLACE POUR  FILSFRERE');

        if egalun (exp) then writeln('C EST EGAL A UN ')
        else writeln ('CE N EST PAS EGAL A UN ')
      end
    else writeln (' PAS ASSEZ DE PLACE POUR FILSFRERE')
  end;

procedure TEST2;

  begin
    repeat
      page (output); gotoxy(0,0);
      writeln ('TAPE L EXPRESSION ');
    until lirexp (ligne,chexp);
    if filsfrere (chexp,exp) then
      begin
        writeln ('ASSEZ DE PLACE POUR  FILSFRERE');

        if egalzero (exp) then writeln('C EST EGAL A UN ')
        else writeln ('CE N EST PAS EGAL A UN ')
      end
    else writeln (' PAS ASSEZ DE PLACE POUR FILSFRERE')
  end;
```

```
procedure TEST3;

  var c : integer;

  begin
    repeat
      page (output); gotoxy(0,0);
      writeln ('TAPE L EXPRESSION NOMBRE ');
    until lirexp (ligne,chexp);
    if filsfrere (chexp,exp) then
      begin
        writeln ('ASSEZ DE PLACE POUR  FILSFRERE');

        c := caracnbre (exp);
        writeln(c);
      end
    else writeln (' PAS ASSEZ DE PLACE POUR FILSFRERE')
  end;

procedure TEST4;

  var e,r,l,i : integer;

  begin
    writeln ('TAPE UN ENTIER DE 3 CHIFFRES MAX ');
    read (e);

    if entiercarac (e,r,l) then
      begin
        writeln (' ASSEZ DE PLACE POUR LA CONVERSION ');
        writeln ('L:',l);
        writeln ('R:',r);
        for i := 1 to l do
          begin
            write (tabcar[r]);
            r := r + 1
          end
      end
    end;


  procedure TEST5;

    var exp1,exp2 : refnoeud;

    begin
      repeat
        page (output); gotoxy(0,0);
        writeln ('TAPE LA PREMIERE EXPRESSION ');
      until lirexp (ligne,chexp);
      if filsfrere (chexp,exp1) then
        begin
          writeln ('ASSEZ DE PLACE POUR  FILSFRERE');
          if (not ordarb(exp1)) then
            writeln(' PAS ASSEZ DE PLACE POUR ORDONNER')
          else
            begin
              repeat
                page (output); gotoxy(0,0);
                writeln ('TAPE LA SECONDE EXPRESSION ');
              until lirexp (ligne,chexp);
              if filsfrere (chexp,exp2) then
```

```
                begin
                  writeln ('ASSEZ DE PLACE POUR  FILSFRERE');
                  if (not ordarb(exp2)) then
                    writeln(' PAS ASSEZ DE PLACE POUR ORDONNER')
                  else
                    if chegal(exp1,exp2) then writeln(' EXP1 = EXP2 ')
                    else writeln ('EXP1 <> EXP2')
                end
              else writeln (' PAS ASSEZ DE PLACE POUR FILSFRERE ');
            end
        end

      else writeln (' PAS ASSEZ DE PLACE POUR FILSFRERE ');

    end;


procedure ECRIRE (exp : refnoeud);

      var i,l,r : integer;

      begin
        page (output);
        if exp < 0 then exp := -exp;
        r := tabnoeud[exp].refch;
        l := tabnoeud[exp].longch;
        for i := 1 to l do
          begin
            write (tabcar[r]);
            r := r + 1
          end;
        writeln (' ');
        wait;
        page (output)
      end;


  procedure TEST7;

    var exp1 : refnoeud;

    begin
      repeat
        page (output); gotoxy(0,0);
        writeln ('TAPE LA PREMIERE EXPRESSION ');
      until lirexp (ligne,chexp);
      if filsfrere (chexp,exp1) then
        begin
          writeln ('ASSEZ DE PLACE POUR  FILSFRERE');
          if (not ordarb(exp1)) then
            writeln(' PAS ASSEZ DE PLACE POUR ORDONNER')
          else ecrire (exp1)
        end

      else writeln (' PAS ASSEZ DE PLACE POUR FILSFRERE ');

    end;
```

```
procedure TEST6;

  var pere1,pere2,crt,c : refnoeud;
      ch : char;
      fin : boolean;



  begin
    repeat
      page (output); gotoxy(0,0);
      writeln ('TAPE L EXPRESSION ');
    until lirexp (ligne,chexp);
    if filsfrere (chexp,pere1) then
      begin
        writeln ('ASSEZ DE PLACE POUR  FILSFRERE');
        if duplicata (pere1,pere2) then
          begin
            writeln ('ASSEZ DE PLACE POUR DUPLICATA ');
            ecrire (pere2);
            crt := pere2;
            fin := false;
            repeat
              writeln(' FILS : I');
              writeln(' FRERE : R');
              writeln (' FIN : F');
              repeat read (ch) until ch in ['I','R','F'];
              case ch of
                 'I' : begin
                         c := tabnoeud[crt].reffi;
                         if c < 0 then c := (-c);
                         if (c = 0) then writeln('BLOQUE 0')
                         else
                           begin
                             crt := c;
                             ecrire (crt)
                           end
                       end;
                 'R' : begin
                         c := tabnoeud[crt].reffr;
                         if c < 0 then c := (-c);
                         if (c = 0) then writeln('BLOQUE 0')
                         else
                           begin
                             crt := c;
                             ecrire (crt)
                           end
                       end;
                 'F' : fin := true
              end;
            until fin
          end
        else writeln (' PAS ASSEZ DE PLACE POUR DUPLICATA ')
      end
    else writeln ('PAS EASSEZ DE PLACE POUR FILSFRERE')
  end;

begin
  premcar := 1;
  dercar := nc;
  premnoeud := 1;
  dernoeud := nn;
```

```
  veux := true;
  ligne := 7;
  repeat
    writeln('EGALUN : 1');
    writeln('EGALZERO : 2');
    writeln('CARACNBRE : 3');
    writeln('ENTIERCARAC : 4');
    writeln('CHEGAL : 5');
    writeln('DUPLICATA : 6');
    read(ch);
    case ch of
      '1' : test1;
      '2' : test2;
      '3' : test3;
      '4' : test4;
      '5' : test5;
      '6' : test6;
      '7' : test7
    end;
    wait;
    page (output);
    writeln('ENCORE TESTER ');
    repeat read (ch) until ch in ['O','N'];
    veux := (ch = 'O')
  until (not veux)
end.

(*-------------------------------------*)
```

```
(*=======================================

            Annexe 4.6.

          UNIT MISEENEVIDENCE

=======================================
```

4.6.1. SPECIFICATIONS

Ce unit comprend l'implementation des fonctions suivantes:
- -DEGREQUOT: degre quotient ou multiplicite d'un facteur dans une somme
- -DEGREQ: multiplicite d'un facteur dans une expression <> somme.
- -MEE: mise en evidence proprement dite
- -EXPOS: construction de puissance
- -PROD: construction de produit
- -DIVISION: mise en evidence d'un facteur a une certaine multiplicite.

```
FUNCTION MEE (EXPD: REFNOEUD; VAR EXPF: REFNOEUD): BOOLEAN;
```

PRCD: Expd est la reference a la racine d'un arbre fils-frere somme.
Expf est une reference dans tabnoeud.

PSCD: Renvoie vrai si il y a eu suffisamment de place pour mettre en evidence et si la mise en evidence est possible sur <expd>;
auquel cas
<expf> = le resultat de la mise en evidence des facteurs communs de <expd>
equi-math(<expd>,<expf>)
(non encore implemente)

Renvoie faux sinon auquel cas
expf = 0 si il n'y a pas eu assez de place pour construire la mise en evidence
expf = -1 si il y a eu assez de place mais la mise en evidence a echoue faute de facteur commun.

COMMENT:
Voir l'algorithme du chapitre V paragraphe 4., auquel on a ajoute la gestion de la place memoire.

IMBRIQUES:
CREERRESTE, CREERFACTCOM, TRTFACT.

FUNCTION CREERESTE : BOOLEAN;

PRCD: Meme que MEE.

PSCD: Renvoie vrai si il ya suffisamment de place pour creer <expr> representant exactement la meme expression que <expd>.

FUNCTION CREERFACTCOM : BOOLEAN;

PRCD: Meme que MEE.

PSCD: Renvoie vrai si il ya suffisamment de place pour creer l'exression qui representera le produit des facteurs communs <expf>. Au debut, on a pas encore trouve de facteur commun donc <expf> = UN.

PROCEDURE TRTFACT(FACTEUR: REFNOEUD);

PRCD: <facteur> est un facteur de <expd>. Tjrsplace = true.

PSCD: Si la base de <facteur> est un facteur commun <fc> de <expr> (c'est a dire que sa multiplicite d <> 0 (degreq), si il y a assez de place pour construire la division de <expr> par <fc> et le produit de <expf> par <fc> alors cette fonction renvoie tjrsplace a vrai.
Elle renvoie tjrsplace a faux sinon.

FUNCTION DEGREQ (DIVID,DIVIS: REFNOEUD): INTEGER;

PRCD: Divid (dividende) et divis (diviseur) sont les references aux des racines d'arbre fils-frere.

PSCD: Renvoie la multiplicite du diviseur en tant que facteur du dividende. Si dividende est une somme alors degreq = 0 sauf si chegal(divid,divis) auquel cas degreq = 1.

IMBRIQUES:
DEGREPU, DEGREMU.

FUNCTION DEGREQUOT(DIVID,DIVIS: REFNOEUD): INTEGER;

PRCD: Divid (dividende) et divis (diviseur) sont les references dans tabnoeud a des racines d'arbre fils-frere.

PSCD: Renvoie la multiplicite du diviseur en tant que facteur du dividende. Si dividende n'est pas une somme alors degrequot = degreq(divid,divis). Sinon degrequot = multiplicite de <divis>

comme facteur commun (v. definition chapitre IV paragraphe 4.3.4.) des termes de <divid>.

```
FUNCTION PARPROPRE(EXP: REFNOEUD): INTEGER;
```

PRCD: Exp est la coordonnee de la racine d'une expression.

PSCD: renvoie le nombre de parentheses propres du noeud reference par exp. Des parentheses sont propres si elles entourent le noeud en question mais ne se retrouvent pas chez le noeud fils.
Exemple:
((a)) a 2 parenthes propres,
(a*b)*(a*c) n'a pas de parenthese propre car se retrouve chez son fils (a*b).

```
FUNCTION EXPOS (VAR PUIS: REFNOEUD; BBASE: REFNOEUD; D: INTEGER):
               BOOLEAN;
```

PRCD: Bbase est la reference dans tabnoeud de la racine d'un arbre fils-frere et puis est une reference dans tabnoeud. D est un entier.

PSCD: Renvoie vrai si il y a assez de place pour construire l'expression puissance <puis> dont la base est l'expression representee par <bbase> et l'exposant est d.
Si <bbase> est l'expression zero alors le resultat est zero.
Si d = 1 alors le resultat est bbase.
Si d = 0 alors le resultat est un sauf si bbase est zero.
Si <bbase> est une puissance alors appliquer la regle suivante a^n*a^m = a^(n+m).
Renvoie faux sinon.

IMBRIQUES:
TRT1,TRT2,CREERFILS, CREERFRERE.

```
FUNCTION CREERFILS: BOOLEAN;
```

PRCD: Fils est la refererence a la racine d'un arbre fils-frere.
Ffils est une reference dans tabnoeud.

PSCD: Renvoie vrai s'il y a assez de place pour contruire <ffils> qui est un arbre fils-frere representant la meme expression que <fils>
Renvoie faux sinon.

```
FUNCTION CREERFRERE: BOOLEAN;
```

PRCD: Frere est la reference a la racine d'un arbre fils-frere, <frere> est le noeud representant l'exposant de la puissance

<bbase>. D est un entier. R, 1 sont des references dans tabcar, ffrere est une reference dans tabnoeud.

PSCD: Renvoie vrai si il y a assez de place pour construire l' expression <ffrere> representant la somme de d avec l'entier represente par <frere>.

FUNCTION TRT1: BOOLEAN;

PRCD: <bbase> n'est pas une puisssance

PSCD: renvoie vrai si creerfils = true, si valeur (ffrere,d) = true (representer la valeur d sous la forme d'un arbre <ffrere>)) et si il y a assez de place pour remplire le champ chaine de caracteres de la racine de l'expression puissance <puis> a construire. renvoie faux sinon.

IMBRIQUES:
CREERBASE, PARENT.

FUNCTION PARENT (EXP: REFNOEUD): IMTEGER;

PRCD: Exp est la racine d'un arbre fils-frere. susceptible de devenir la base d'une puissance.

PSCD: Renvoie 2 si il y a besoin d'entourer <exp> par des parentheses dans la chaine representant une puissance dont <exp> est la base
Renvoie 0 sinon.

FUNCTION CREERBASE: BOOLEAN;

PRCD: <bbase> n'est pas une puissance et bbase = fils. Il y a eu suffisamment de place pour construire la base <ffils> et pour construire l'exposant <ffrere>.

PSCD: Renvoie vrai si il y a suffisammnt de place pour garnir le champs chaine de caracteres du noeud (puis) racine de la puissance a construire. Ce champs est la concatenation des chaines de <ffils> et de <ffrere> et du symbole " ^" et des parentheses eventuellement necessaires pour entourer la base.
Renvoie faux sinon.

FUNCTION TRT2: BOOLEAN;

PRCD: <bbase> est une puissance.

PSCD: renvoie vrai si il y a assez de place pour construire la puissance dont la base est <ffils> et dont l'exposant est <ffrere>. <ffils> represente la meme expression que le fils

de <bbase> et <ffrere> represente l'expression valeur entiere somme de l'exposant de <bbase> et de d.

IMBRIQUES:
CREERBASE.

FUNCTION CREERBASE: BOOLEAN;

PRCD: <bbase> est une puissance et fils = la reference au fils de <bbase>.
Il y a eu suffisamment de place pour construire la base <ffils> et pour construire l'exposant <ffrere>.

PSCD: renvoie vrai si il ya suffisamment de place pour garnir le champs chaine de la racine de la puissace a construire (puis).
Ce champs est la concatenation des champs de <ffils> et de <ffrere>, du signe "^" et d'eventuelles parentheses qui entourent la base.
renvoie faux sinon.

FUNCTION PROD (VAR EXP: REFNOEUD; FACT1, FACT2: REFNOEUD):BOOLEAN;

PRCD: Fact1 et fact2 sont les references aux racines de deux arbres fils-freres. Exp est une reference dans tabnoeud.

PSCD: Renvoie vrai si il y a assez de place pour construire l'expression representant le produit de <fact1> et de <fact2>.

COMMENT:
Pour les cas particuliers, ou <fact1> et (ou) <fact2> = un alors appliquer la propriete de neutre pour la multiplication.
Suivant que fact1 et fact2 sont des produits ou non adapter la construction de l'arbre produit.
Rem: a partir de <fact1> et de <fact2>, on construit les arbres <ffact1> et <ffact2> representant exactement les memes expressions.
C'est en realite ffact1 et ffact2 qui entreront dans la construction du produit.

IMBRIQUES:
CREERFACT (s'occupe de la duplication des facteurs <fact1> et <fact2>,
(CPUN (s'occupe des cas particuliers, facteur(s) = un),
PROD1 ET PROD2 (decoupe due au compilateur),
CREERPERE.

FUNCTION CREERPERE (VAR PERE: REFNOEUD; F1,F2 : REFNOEUD): BOOLEAN;

PRCD: F1 et f2 sont les references aux racines de deux arbre fils-freres.

PSCD: Renvoie vrai si il y a assez de place pour reserver le noeud racine du produit de <f1> par <f2> et si il ya assez de place pour garnir le champs chaine de caracteres du noeud reserve.
Renvoie faux sinon.

IMBRIQUES:
OUVPAR, RECOP1, RECOP2, FERMEPAR : utilitaires de remplissage du champs chaine de caracteres.
PARENT: renvoie le nombre de parenthese a inserer dans la chaine de caracteres.

FUNCTION DIVISION (VAR DIVID: REFNOEUD; DIVIS: REFNOEUD; D: INTEGER): BOOLEAN;

PRCD: Divid et divis sont les references aux noeuds de deux arbres fils-frere. Divis est un facteur commun aux terme de <divid>. Divis a une multiplicite d dans <divid>.

PSCD: Renvoie vrai si il y assez de place pour mettre en evidence <divis> a la multiplicite d; auquel cas divid est la reference a la racine du resultat.

COMMENT:
Pour chaque terme <t> de <divid>, diviser <t> par <divis> a la multplicite d (QUOT (t,divis,d)).

IMBRIQUES:
QUOT,RECOPIER, CREERPL.

FUNCTION CREERPL (PERE,PREM,DER: REFNOEUD): BOOLEAN;

PRCD: Pere est un erference dans tabnoeud. Prem, der sont les references aux racines des premier et dernier termes de la somme <pere> a creer.

PSCD: Renvoie vrai si il y a assez de place pour creer <pere>, arbre fils-frere somme de premier terme <prem> et de dernier terme <der>.

PROCEDURE RECOPIER (VAR R1, R2: INTEGER; L: INTEGER);

PRCD: L est un entier (longueur). R1, r2 sont des references dans tabcar.

PRSD: R1 et r2 sont incrementes de 1. La zone de tabcar commencant en r1 (avant incrementation) et terminant en (r1+l-1) a ete recopiee a partie de r2 (avant incrementation).

FUNCTION QUOT (VAR DIVID: REFNOEUD; DIVIS: REFNOEUD; VAR D: INTEGER);

PRCD: Divid (dividende) et divis (diviseur) sont les references a deux arbres fils-freres. <Divis> est un facteur de <divid>.

PSCD: Soient d1 la multiplicite de <divis> comme facteur de <divid>, si d<=d1 alors d2 = 0 et d3 = d sinon d2 = d-d1 et d3 = d1. Renvoie vrai si il y a suffisamment de place pour construire

le quotient de <divid> par <divis> a la multiplicite d3.
D vaut d2.
Renvoi faux sinon.

IMBRIQUES:
QUOTPU: realise quot pour un dividende = puissance
QUOTMU: realise quot pour un dividende = produit
CREERMO.

```
FUNCTION QUOTPU (VAR DIVID: REFNOEUD; DIVIS: REFNOEUD; VAR D: INTEGER)
                :BOOLEAN;
```

PRCD: Divid et divid sont les references aux racines de deux arbres fils-frere. <divid> est une puissance.

PSCD: Renvoie vrai si il y a assez de place. Diminue d de la multiplicite de <divis> dans <divid>. <divid> est le resultat de la division.
Renvoie faux sinon.

IMBRIQUES:
CREERPU.

```
FUNCTION CREERPU (PERE, FILS,FRERE: REFNOEUD; VAR D: INTEGER): BOOLEAN;
```

PRCD: Fils et frere sont les references a deux arbre fils-freres. Pere est une reference dans tabnoeud.

PSCD: Renvoie vrai si il y a assez de place pour creer la puissance <pere> de fils <fils> et d' exposant <frere>
Renvoie faux sinon.

IMBRIQUES:
NBREPAR.

```
FUNCTION QUOTMU (VAR DIVID: REFNOEUD; DIVIS: REFNOEUD; VAR D: INTEGER)
                :BOOLEAN;
```

PRCD: Divid et divid sont les references aux racines de deux arbres fils-frere. <divid> est un produit.

PSCD: Renvoie vrai si il y a assez de place pour realiser la division de <divid> par <divis> a la multiplicite d. D est diminue de la multiplicite de <divis>. <divid> est le resultat de la division.
Renvoie faux sinon.

IMBRIQUES:
CREERMU, QUOTFACT.

```
FUNCTION CREERMU (PERE, PREM, DER: REFNOEUD): BOOLEAN;
```

```
    PRCD: Pere, prem, der sont les references aux racines de trois
          arbre fils-freres.

    PSCD: Renvoie vrai si il y a assez de place pour creer le produit
          <pere> dont le premier facteur est <prem> et le dernier <der>.
          Tient compte des regles de calculs sur l'element neutre (un)
          pour la multiplication.

     IMBRIQUES:
       ELIMUN: elimine les facteurs = un apres division,
       NBREPAR: renvoie le nombre de parentheses necessaires a un facteur
                pour etre insere dans la chaine de caracteres representant
                un produit.

   ------------------------------

FUNCTION CREERMO (PERE,FILS: REFNOEUD): BOOLEAN;
-------------------------------------------------

   PRCD: Pere, fils sont les references aux racines de deux arbres fils-
         freres.

   PSCD: Renvoie vrai si il y a assez de place pour creer le moins
         <pere> = -<fils>.

   ------------------------------
-----------------------------------*)
```

```
(*
4.6.2.CODE
      ----*)


(*$S++*)
unit MISEENEVIDENCE;
interface
  uses applestuff,
       (*$U #5:GLOBAL.CODE*) global,
       (*$U #5:UT.CODE*) utilitaires;
  function MEE (expd : refnoeud; var expf : refnoeud) : boolean;
  function DEGREQUOT (divid,divis : refnoeud) : integer;
  function EXPOS (var puis : refnoeud; bbase : refnoeud; d : integer)
                 : boolean;
  function PROD (var exp : refnoeud; fact1,fact2 : refnoeud) : boolean;
  function DIVISION (var divid : refnoeud;divis : refnoeud;d : integer)
           : boolean;
implementation

  function DEGREQ (divid,divis : refnoeud) : integer;


    function DEGREPU (divid,divis : refnoeud) : integer;

      var bbase1,bbase2,expo1,expo2,e1,e2 : integer;

      begin (* DEGREPU *)
        bbase1 := tabnoeud[divid].reffi;
        expo1 := tabnoeud[bbase1].reffr;
        if tabnoeud[divis].tn = pu then
          begin
            bbase2 := tabnoeud[divis].reffi;
            expo2 := tabnoeud[bbase2].reffr;

            if chegal(bbase1,bbase2) then
              if chegal (expo1,expo2) then degrepu := 1
              else
                begin
                  e1 := caracnbre (expo1);
                  e2 := caracnbre (expo2);
                  if e1 < e2 then degrepu := 0
                  else degrepu := e1 div e2
                end
            else
              if chegal(bbase1,divis) then degrepu := caracnbre(expo1)
              else degrepu := 0

          end
        else
          if chegal(bbase1,divis) then degrepu := caracnbre(expo1)
          else degrepu := 0;
    end; (*DEGREPU*)
```

```
  function DEGREMU (divid,divis : refnoeud) : integer;

    var d,fact : integer;


    function DEGREFACT (divid,divis : refnoeud) : integer;

      begin (* DEGREFACT *)
        degrefact := degreq (divid,divis);
      end; (* DEGREFACT*)

    begin (* DEGREMU*)
      d := 0;
      fact := tabnoeud[divid].reffi;
      while (fact >0) do
        begin
          d := d + degrefact(fact,divis);
          fact := tabnoeud[fact].reffr
        end;
      degremu := d;
    end;(* DEGREMU *)


  begin (* Degreq *)
    if chegal (divid,divis) then degreq := 1
    else
      case tabnoeud [divid].tn of

        pl,val,vari : degreq := 0;

        mo : degreq := degreq (tabnoeud[divid].reffi,divis);

        mu : degreq := degremu (divid,divis);

        pu : degreq := degrepu (divid,divis)
      end;
  end; (* Degreq *)


function DEGREQUOT;

  var d,enfcrt,dmin : integer;
      divisible : boolean;

  begin (*DEGREQUOT*)

    case tabnoeud[divid].tn of

      pu,mo,val,vari,mu : degrequot := degreq(divid,divis);

      pl : begin
             if chegal (divid,divis) then degrequot := 1
             else
               begin
                 enfcrt := tabnoeud[divid].reffi;
                 d := degreq(enfcrt,divis);
                 if d = 0 then degrequot := 0
                 else
                   begin
                     enfcrt := tabnoeud[enfcrt].reffr;
                     divisible := true;
                     dmin := d;
```

```
                    while ((enfcrt > 0) and divisible) do
                      begin
                        d := degreq (enfcrt,divis);
                        if d = 0 then divisible := false;
                        if dmin > d then dmin := d;
                        enfcrt := tabnoeud[enfcrt].reffr
                      end;
                    degrequot := dmin
                  end
              end
          end
      end;
    end; (*DEGREQUOT*)

  function PARPROPRE (exp : refnoeud) : integer;

    var r1,r2,np1,np2 : integer;

    begin
      r1 := tabnoeud[exp].refch;
      np1 := 0;
      while (tabcar[r1] in ['(',' ']) do
        begin
          if (tabcar[r1] = '(') then np1 := np1 + 1;r1 := r1 + 1
        end;
      if (tabnoeud[exp].tn in [val,vari]) then parpropre := np1
      else
        begin
          r2 := tabnoeud[tabnoeud[exp].reffi].refch;
          np2 := 0;
          while (tabcar[r2] in ['(',' ']) do
            begin
              if (tabcar[r2] = '(') then np2 := np2 + 1;r2 := r2 + 1
            end;
          if ((np1 - np2) <= 0) then parpropre := 0
          else parpropre := (np1 - np2)
        end
    end;


  function EXPOS;

    var fils,ffils,frere,ffrere,r,l : integer;

    function CREERFILS : boolean;

      begin
        creerfils := duplicata (fils,ffils);
      end;

    function CREERFRERE : boolean;

      begin
        if ((not entiercarac (d*caracnbre(frere),r,l)) or (not place)) then
          creerfrere := false
        else
          begin
            creerfrere := true;
            ffrere := premnoeud;premnoeud := premnoeud + 1;
            tabnoeud[ffrere] := tabnoeud[frere];
```

```
          tabnoeud[ffrere].refch := r;
          tabnoeud[ffrere].longch := 1
        end;
    end;


  function TRT1 : boolean;

   function PARENT (exp : refnoeud) : integer;

     begin
       if (parpropre (exp) <> 0) then parent := 0
       else
         if (tabnoeud[exp].tn in [vari,val]) then parent := 2
         else parent := 0
       end;

   function CREERBASE : boolean;

     var np,lt,r1,r2,i : integer;
         t : typenoeud;

     begin
       t := tabnoeud[fils].tn;
       np := parent (fils);
       lt := np + tabnoeud[ffils].longch + tabnoeud[ffrere].longch + 1;
       if ((not place) or ((premcar + lt) > dercar)) then
         creerbase := false
       else
         begin
           creerbase := true;
           puis := premnoeud;premnoeud := premnoeud + 1;
           tabnoeud[puis] := tabnoeud[bbase];
           tabnoeud[puis].longch := lt;
           tabnoeud[puis].refch := premcar;
           r2 := premcar;
           if np > 0 then
             begin
               tabcar[r2] := '(';
               r2 := r2 + 1; premcar := premcar + 1
             end;
           r1 := tabnoeud[ffils].refch;
           for i := 1 to tabnoeud[ffils].longch do
             begin
               tabcar[r2] := tabcar[r1]; r2 := r2 + 1;
               r1 := r1 + 1; premcar := premcar + 1
             end;
           if np > 0 then
             begin
               tabcar[r2] := ')';
               r2 := r2 + 1; premcar := premcar + 1
             end;
           tabcar[r2] := '^'; r2 := r2 + 1; premcar := premcar + 1;
           r1 := tabnoeud[ffrere].refch;
           for i := 1 to tabnoeud[ffrere].longch do
             begin
               tabcar[r2] := tabcar[r1]; r2 := r2 + 1;
               r1 := r1 + 1; premcar := premcar + 1
             end;
           if np > 0 then
             begin
               tabnoeud[ffils].longch := tabnoeud[ffils].longch + 1;
```

```
            tabnoeud[ffils].refch := tabnoeud[puis].refch + 1
          end
      end
    end;

    begin (* Trt1 *)
      fils := bbase;
      if ((not creerfils) or (not valeur (ffrere,d)) or (not creerbase))
         then trt1 := false
      else
        begin
          trt1 := true;
          tabnoeud[puis].reffr := 0;
          tabnoeud[puis].reffi := ffils;
          tabnoeud[ffils].reffr:=ffrere;
          tabnoeud[ffrere].reffr:=-puis;
          tabnoeud[ffils].ordrinit:=1;
          tabnoeud[ffrere].ordrinit:=2
        end
    end; (* Trt1 *)

function TRT2 : boolean;


  function CREERBASE : boolean;

  var lt,r1,r2,i : integer;

  begin
    lt := tabnoeud[ffils].longch + tabnoeud[ffrere].longch + 1;
    if ((not place) or ((premcar + lt) > dercar)) then
      creerbase := false
    else
      begin
        creerbase := true;
        puis := premnoeud;premnoeud := premnoeud + 1;
        tabnoeud[puis] := tabnoeud[bbase];
        tabnoeud[puis].longch := lt;
        tabnoeud[puis].refch := premcar;
        r2 := premcar;
        r1 := tabnoeud[ffils].refch;
        for i := 1 to tabnoeud[ffils].longch do
          begin
            tabcar[r2] := tabcar[r1]; r2 := r2 + 1;
            r1 := r1 + 1; premcar := premcar + 1
          end;
        tabcar[r2] := '^'; r2 := r2 + 1; premcar := premcar + 1;
        r1 := tabnoeud[ffrere].refch;
        for i := 1 to tabnoeud[ffrere].longch do
          begin
            tabcar[r2] := tabcar[r1]; r2 := r2 + 1;
            r1 := r1 + 1; premcar := premcar + 1
          end
      end;
    end;

    begin (* Trt2 *)
      fils := tabnoeud[bbase].reffi;
      frere := tabnoeud[fils].reffr;
      if ((not creerfils) or (not creerfrere) or (not creerbase)) then
        trt2 := false
      else
```

```
          begin
            trt2 := true;
            tabnoeud[puis].reffr := 0;
            tabnoeud[puis].reffi := ffils;
            tabnoeud[ffils].reffr:=ffrere;
            tabnoeud[ffrere].reffr:=-puis;
            tabnoeud[ffils].ordrinit:=1;
            tabnoeud[ffrere].ordrinit:=2
          end;
      end;  (* Trt2 *)

  begin (* Expos *)
    if d = 0 then
      if egalzero (bbase) then expos := zero(puis)
      else expos := un(puis)
    else
      if d = 1 then expos := duplicata(bbase,puis)
      else
        if egalun (bbase) then expos := un(puis)
        else
          if tabnoeud[bbase].tn = pu then expos := trt2
          else expos := trt1;
  end; (* Expos *)

function PROD;

  var t1,t2 : typenoeud;
      ffact1,ffact2,enfcrt,prec : refnoeud;

  function CREERFACT (exp : refnoeud; var eexp : refnoeud) : boolean;

    begin
      creerfact := duplicata (exp,eexp);
    end;

  function CREERPERE (var pere : refnoeud; f1,f2 : refnoeud) : boolean;

    var np,lt,r1,r2 : integer;

    procedure OUVPAR;

      begin
        tabcar[r2] := '('; r2 := r2+ 1;
        premcar  := premcar + 1
      end;

    procedure FERMPAR;

      begin
        tabcar[r2] := ')'; r2 := r2+ 1;
        premcar := premcar + 1
      end;

    procedure RECOP2;

      var i : integer;

      begin
        r1 := tabnoeud[fact2].refch;
        for i := 1 to tabnoeud[fact2].longch do
          begin
            tabcar[r2] := tabcar[r1];
```

```
          r1 := r1 + 1; r2 := r2 + 1;
          premcar := premcar + 1
        end
    end;

  procedure ETOILE;

    var i : integer;

    begin
      tabcar[r2] := '*'; r2 := r2 + 1;
      premcar := premcar + 1
    end;
 procedure RECOP1;

    var i : integer;

    begin
      r1 := tabnoeud[fact1].refch;
      for i := 1 to tabnoeud[fact1].longch do
        begin
          tabcar[r2] := tabcar[r1];
          r1 := r1 + 1; r2 := r2 + 1;
          premcar := premcar + 1
        end
    end;


function PARENT (exp :refnoeud) : integer;

  begin
    if (parpropre (exp) <> 0 ) then parent := 0
    else
      if (tabnoeud[exp].tn in [pl,mo]) then parent := 2
      else parent := 0
  end;



 begin (* Creerpere *)
    np := 0;
    np := (parent(f1)+parent(f2));
    lt := (np + tabnoeud[f1].longch +tabnoeud[f2].longch + 1);
    if ((not place) or ((premcar + lt) > dercar)) then
      creerpere := false
    else
      begin
        creerpere := true;
        pere := premnoeud;premnoeud := premnoeud + 1;
        tabnoeud[pere].longch := lt;
        r2 := premcar;
        tabnoeud[pere].refch := premcar;
        case np of
          0 : begin
                recop1; etoile; recop2
              end;

          2 : begin
                if (parent(f1) = 2) then
                  begin
                    ouvpar; recop1; fermpar; etoile; recop2;
                    tabnoeud[f1].refch := tabnoeud[pere].refch;
```

```
                    tabnoeud[f1].longch := tabnoeud[f1].longch + 2
                  end
                else
                  begin
                    recop1; etoile;tabnoeud[f2].refch := r2; ouvpar;
                     recop2; fermpar;
                    tabnoeud[f2].longch := tabnoeud[f2].longch + 2
                  end
              end;
          4 : begin
                ouvpar; recop1; fermpar; etoile;
                tabnoeud[f2].refch := r2;
                ouvpar;recop2; fermpar;
                tabnoeud[f1].refch := tabnoeud[pere].refch;
                tabnoeud[f1].longch := tabnoeud[f1].longch + 2;
                tabnoeud[f2].longch := tabnoeud[f2].longch + 2
              end
        end;
      tabnoeud[exp].tn := mu;
      tabnoeud[exp].reffr := 0;
      tabnoeud[exp].ordrinit := 1;
      tabnoeud[exp].refsol := 0;
      tabnoeud[exp].methode := 0;
      tabnoeud[exp].etape := 0;
      tabnoeud[exp].refcan := false
    end;
  end;(* Creerpere *)

procedure PROD1;
    begin
      prec := tabnoeud[ffact1].reffi;
      enfcrt := tabnoeud[prec].reffr;
      tabnoeud[exp].reffi:= prec;
      while (enfcrt > 0) do
        begin
          prec := enfcrt;
          enfcrt := tabnoeud[enfcrt].reffr
        end;
      if (t2 = mu) then
        begin
          enfcrt := tabnoeud[ffact2].reffi;
          tabnoeud[enfcrt].ordrinit := tabnoeud[prec].ordrinit + 1;
          tabnoeud[prec].reffr := enfcrt;
          prec := enfcrt;
          enfcrt := tabnoeud[enfcrt].reffr;
          while (enfcrt > 0) do
            begin
              tabnoeud[enfcrt].ordrinit := tabnoeud[prec].ordrinit + 1;
              prec := enfcrt;
              enfcrt := tabnoeud[enfcrt].reffr
            end;
          tabnoeud[prec].reffr := -exp
        end
      else
        begin
          tabnoeud[prec].reffr := ffact2;
          tabnoeud[ffact2].reffr := -exp;
        end;
    end;
```

```
procedure PROD2;
  begin
    tabnoeud[exp].reffi:= ffact1;
    if (t2 = mu) then
      begin
        tabnoeud[ffact1].ordrinit := 1;prec := tabnoeud[ffact2].reffi;
        tabnoeud[prec].ordrinit := 2;
        enfcrt := tabnoeud[prec].reffr;
        tabnoeud[ffact1].reffr := tabnoeud[ffact2].reffi;
        while (enfcrt > 0) do
          begin
            tabnoeud [enfcrt].ordrinit := tabnoeud[prec].ordrinit + 1;
            prec := enfcrt;
            enfcrt := tabnoeud[enfcrt].reffr
          end;
        tabnoeud[prec].reffr := -exp
      end
    else
      begin
        tabnoeud[ffact1].reffr := ffact2;
        tabnoeud[ffact1].ordrinit := 1;
        tabnoeud[ffact2].reffr := -exp;
        tabnoeud[ffact2].ordrinit := 2
      end;
  end;


procedure CPUN;

  begin
    if egalun(fact1) then
      if egalun(fact2) then prod := un(exp)
      else
        if (not creerfact(fact2,ffact2)) then prod := false
        else
          begin
            prod := true; exp := ffact2; tabnoeud[exp].reffr := 0
          end
    else
      if (not creerfact(fact1,ffact1)) then prod := false
      else
        begin
          prod := true; exp := ffact1; tabnoeud[exp].reffr := 0
        end
  end;


begin (* Prod *)
  t1 := tabnoeud [fact1].tn;
  t2 := tabnoeud[fact2].tn;
  if (egalun(fact1) or egalun(fact2)) then cpun
  else

    if (not creerfact (fact1,ffact1)) then prod := false
    else
      if (not creerfact (fact2,ffact2)) then prod := false
      else
        if (not creerpere (exp,ffact1,ffact2)) then  prod := false
        else
          begin
            prod := true;
            tabnoeud[exp].reffr:=0;
```

```
                if (t1 = mu) then prod1
                else prod2
              end;
  end; (*Prod *)

(*$I #5:MEE2.TEXT*)

(*-----------------------------------------*)
```

```
function DIVISION;

var enfcrt,der,prec,prem,dd : integer;
    tjrsplace : boolean;

procedure RECOPIER (var r1,r2 : integer;l : integer);
  var i : integer;
  begin
    for i := 1 to l do
      begin
        tabcar[r2] := tabcar[r1];
        r2 := r2 + 1;r1 := r1 + 1
      end
  end;
procedure SIGNE (var r : integer;c : char);
  begin
    tabcar[r] := c; r := r + 1
  end;

function QUOT (var divid : refnoeud;divis : refnoeud;var d : integer)
         : boolean;

  var expi : refnoeud;

function QUOTPU (var divid : refnoeud;divis : refnoeud;var d : integer)
     : boolean;

  var expo1,expo2,e1,e2,e,r,bbase1,bbase2 : integer;

  function CREERPU (pere,fils,frere : refnoeud;e : integer) : boolean;

    var r,l,r1,r2,np : integer;

    function NBREPAR (exp : refnoeud) : integer;

      var fils : integer;

      begin
        if (tabnoeud[exp].tn in [vari,val,mu,pu]) then nbrepar := 0
        else
          if (parpropre (exp) = 0) then nbrepar := 2
          else nbrepar := 0
      end;


    begin (* CREERPU *)
      if ( e = 0) then creerpu := un (divid)
      else
        if (e = 1) then
          begin
            divid := fils; tabnoeud[fils].reffr := 0;
            creerpu := true;
          end
        else
```

```
    if (not entiercarac(e,r,l)) then creerpu := false
    else
      begin
        tabnoeud[pere].reffi := fils;
        tabnoeud[fils].reffr := frere;
        tabnoeud[frere].reffr := -pere;
        tabnoeud[frere].refch := r;
        tabnoeud[frere].longch := 1;
        np := nbrepar (fils);
        l := l + tabnoeud[fils].longch + 1 + np;
        if ((premcar + l ) > dercar) then creerpu := false
        else
          begin
            creerpu := true;
            tabnoeud[pere].refch := premcar;
            r2 := premcar;
            r1 := tabnoeud[fils].refch;
            if np > 0 then signe (r2,'(');
            recopier (r1,r2,tabnoeud[fils].longch);
            if np > 0 then signe (r2,')');
            signe (r2,'^');
            r1 := tabnoeud[frere].refch;
            recopier (r1,r2,tabnoeud[frere].longch);
            premcar := premcar + 1;
            tabnoeud[pere].longch := l
          end
      end;
  end; (* CREERPU *)


begin (*QUOTPU*)
  bbase1 := tabnoeud[divid].reffi;
  expo1 := tabnoeud[bbase1].reffr;
  if tabnoeud[divis].tn = pu then
    begin
      bbase2 := tabnoeud[divis].reffi;
      expo2 := tabnoeud[bbase2].reffr;
      if chegal (bbase1,bbase2) then
        if chegal (expo1,expo2) then
          begin
            d := d - 1; quotpu := un (divid)
          end
        else
          begin
            e1 := caracnbre (expo1);
            e2 := caracnbre (expo2);
            if e1 > e2 then
              begin
                e := e1 div e2;
                r := e1 mod e2;
                if d >= e then
                  begin
                    d := d - e;
                    quotpu := creerpu(divid,bbase1,expo1,r)
                  end
                else
                  begin
                    e := (e - d) * e2 + r;
                    d := 0;
                    quotpu := creerpu (divid,bbase1,expo1,e)
                  end
```

```
            end
          else quotpu := true
        end
      else
        if chegal (bbase1,divis) then
          begin
            e1 := caracnbre (expo1);
            if d>e1 then
              begin
                d := d - e1;
                quotpu := un (divid)
              end
            else
              begin
                quotpu := creerpu (divid,bbase1,expo1,e1 - d);
                d := 0
              end
          end
        else quotpu := true

    end
  else
    if chegal (bbase1,divis) then
      begin
        e1 := caracnbre (expo1);
        if d>e1 then
          begin
            d := d - e1;
            quotpu := un (divid)
          end
        else
          begin
            quotpu := creerpu (divid,bbase1,expo1,e1 - d);
            d := 0
          end
      end
    else quotpu := true;
end; (* QUOTPU *)

function QUOTMU (var divid : refnoeud;divis : refnoeud;var d : integer)
      : boolean;

  var enfcrt,prem,der,prec,np : integer;
      tjrsplace : boolean;


  function CREERMU (pere,prem,der : refnoeud) : boolean;

    var l,lt,enfcrt,r1,r2 : integer;
        tjrsplace : boolean;

    procedure ELIMUN (var prem,der :refnoeud);

      var enfcrt : integer;
          fin : boolean;

      begin
        fin := false;
        while (not fin) do
          begin
            if (prem < 0) then fin := true
```

```
            else
              if egalun (prem) then prem := tabnoeud[prem].reffr
              else fin := true
          end;
        if (prem > 0) then
          begin
            der := prem; enfcrt := tabnoeud[der].reffr;
            fin := false;
            while (not fin) do
              begin
                if (enfcrt < 0) then fin := true
                else
                  begin
                    if (not egalun(enfcrt)) then
                      begin
                        tabnoeud[der].reffr := enfcrt;
                        der := enfcrt
                      end;
                    enfcrt := tabnoeud[enfcrt].reffr
                  end
              end
          end;

function NBREPAR (exp : refnoeud) : integer;

  var fils : integer;

  begin
    if (tabnoeud[exp].tn in [vari,val,mu,pu]) then nbrepar := 0
    else
      if (parpropre (exp) = 0) then nbrepar := 2
      else nbrepar := 0
  end;

    begin (*CREERMU *)
      elimun (prem,der);
      if (prem <0) then creermu := un ( divid)
      else
        if (prem = der) then
          begin
            divid := prem;
            creermu := true
          end
        else

      begin

      tabnoeud[pere].reffi := prem;
      tabnoeud[der].reffr := - pere;
      tabnoeud[pere].reffr := 0;
      enfcrt := prem;
      np := nbrepar (enfcrt);
      l := tabnoeud[enfcrt].longch + np;
      if ((premcar + l) > dercar) then creermu := false
      else
        begin
          tabnoeud[pere].refch := premcar;
          r2 := premcar; r1 := tabnoeud[enfcrt].refch;
          if np > 0 then signe (r2,'(');
          recopier (r1,r2,tabnoeud[enfcrt].longch);
          if np > 0 then signe (r2,')');
```

```
        premcar := premcar + 1;
        lt := 1;
        enfcrt := tabnoeud[enfcrt].reffr;

        tjrsplace := true;
        while ((enfcrt >0) and tjrsplace) do
          begin
            np := nbrepar (enfcrt);
            l := tabnoeud [enfcrt].longch + np + 1;
            lt := lt + 1;
            if ((premcar + 1) > dercar) then tjrsplace := false
            else
              begin
                signe (r2,'*');
                if np > 0 then signe (r2,'(');
                r1 := tabnoeud[enfcrt].refch;
                recopier (r1,r2,tabnoeud[enfcrt].longch);
                if np >0 then signe (r2,')');
                premcar := premcar + 1
              end;
            enfcrt := tabnoeud[enfcrt].reffr
          end;
        if(not tjrsplace) then creermu := false
        else
          begin
            creermu := true;
            tabnoeud[pere].longch := lt
          end
      end

  end
  end; (* CREERMU *)



function QUOTFACT (var divid : refnoeud;divis : refnoeud;
                   var d : integer): boolean;

  begin (*QUOTFACT*)
    quotfact := quot (divid,divis,d);
  end; (* QUOTFACT *)


begin (*QUOTMU*)
  der := tabnoeud[divid].reffi;
  enfcrt := tabnoeud[der].reffr;
  prem := der;
  if (not quotfact (prem,divis,d)) then quotmu := false
  else
    if d = 0 then
      begin
        tabnoeud[prem].reffr := enfcrt;
        while (enfcrt > 0) do
          begin
            der := enfcrt;
            enfcrt := tabnoeud[enfcrt].reffr
          end;
        quotmu := creermu (divid,prem,der)
      end
    else
      begin
```

```
          tjrsplace := true;der := prem;
          while(tjrsplace and (d > 0) and (enfcrt > 0)) do
            begin
              prec := enfcrt;
              enfcrt := tabnoeud [enfcrt].reffr;
              if quotfact (prec,divis,d) then
                begin
                  tabnoeud[der].reffr := prec;
                  der := prec;
                  prec := enfcrt
                end
              else tjrsplace := false
            end;
          if not tjrsplace then quotmu := false
          else
            begin
              if (d = 0) then
                begin
                  tabnoeud[der].reffr := enfcrt;
                  while (enfcrt >0) do
                    begin
                      der := enfcrt;
                      enfcrt := tabnoeud[enfcrt].reffr;
                    end
                end;
              quotmu := creermu (divid,prem,der)
            end
        end;
    end; (* QUOTMU *)


  function CREERMO (pere,fils : refnoeud) : boolean;

    var np,lt,r1,r2 : integer;

    begin (* CREERMO *)
      if tabnoeud[fils].tn in [mo,pl] then np := 2
      else np := 0;
      lt := tabnoeud[fils].longch + np + 1;
      if ((premcar + lt) > dercar) then creermo := false
      else
        begin
          creermo := true;
          tabnoeud[pere].reffi := fils;
          tabnoeud[fils].reffr := -pere;
          tabnoeud[pere].reffr := 0;
          tabnoeud[pere].refch := premcar;
          tabnoeud[pere].longch := lt;
          r1 := tabnoeud[fils].refch;
          r2 := premcar;
          signe(r2,'-');
          if np > 0 then signe (r2,'(');
          recopier (r1,r2,tabnoeud[fils].longch);
          if np > 0 then signe (r2,')');
          premcar := premcar + lt
        end;
    end; (*CREERMO *)


  begin (* QUOT *)
    if chegal(divid,divis) then
      begin
```

```
        d := d - 1; quot := un (divid)
      end
    else
      case tabnoeud[divid].tn of

        (* DIVID,D RESTENT INCHANGES DANS LES CAS DE VARI,VAL,PL*)

        vari,val,pl : quot := true;
        mo : begin
               expi := tabnoeud[divid].reffi;
               if (not quot (expi,divis,d)) then quot := false
               else quot := creermo (divid,expi)
             end;

        mu : quot := quotmu (divid,divis,d);

        pu : quot := quotpu (divid,divis,d)
      end;
  end; (*QUOT*)


function CREERPL (pere,prem,der : integer) : boolean;

  var l,lt,enfcrt,r1,r2,np : integer;
      tjrsplace : boolean;
  begin (*CREERPL *)
    tabnoeud[pere].reffi := prem;
    tabnoeud[der].reffr := - pere;
    tabnoeud[pere].reffr := 0;
    enfcrt := prem;
    l := tabnoeud[enfcrt].longch;
    if ((premcar + l) > dercar) then creerpl := false
    else
      begin
        tabnoeud[pere].refch := premcar;
        r2 := premcar; r1 := tabnoeud[enfcrt].refch;
        recopier (r1,r2,l);
        premcar := premcar + l;
        lt := l;
        enfcrt := tabnoeud[enfcrt].reffr;

        tjrsplace := true;
        while ((enfcrt >0) and tjrsplace) do
          begin
            if tabnoeud[enfcrt].tn <> mo then
              begin
                if ((premcar + 1) > dercar) then tjrsplace := false
                else
                  begin
                    lt := lt +1;
                    signe(r2,'+'); premcar := premcar + 1
                  end
              end;
            if tjrsplace then
              begin
                l := tabnoeud [enfcrt].longch;
                lt := lt + l;
                if ((premcar + l) > dercar) then tjrsplace := false
                else
                  begin
                    r1 := tabnoeud[enfcrt].refch;
                    recopier (r1,r2,l); premcar := premcar + l
```

```
                end;
              enfcrt := tabnoeud[enfcrt].reffr
            end
        end;
      if(not tjrsplace) then creerpl := false
      else
        begin
          creerpl := true;
          tabnoeud[pere].longch := lt
        end
    end;
end; (* CREERPL *)


begin (* DIVISION *)

  if (d = 0) then division := zero (divid)
  else

  case tabnoeud[divid].tn of

    pu,mo,val,vari,mu : division := quot (divid,divis,d);

    pl : begin
           if chegal (divid,divis) then division := un(divid)
           else
             begin
               dd := d;
               der := tabnoeud[divid].reffi;
               enfcrt := tabnoeud[der].reffr;
               prem := der;
               if (not quot (prem,divis,d)) then division := false
               else
                 begin
                   tjrsplace := true;der := prem;
                   while(tjrsplace and (enfcrt > 0)) do
                     begin
                       d := dd;
                       prec := enfcrt;
                       enfcrt := tabnoeud [enfcrt].reffr;
                       if quot (prec,divis,d) then
                         begin
                           tabnoeud[der].reffr := prec;
                           der := prec;
                           prec := enfcrt
                         end
                       else tjrsplace := false
                     end;
                   if not tjrsplace then division := false
                   else
                     division := creerpl (divid,prem,der)
                 end
             end
      end
    end;
end; (* DIVISION *)
```

```
function MEE;

  var d : integer;
      expi,terme,expb,expr,fc,facteur : refnoeud;
      tjrsplace : boolean;


function CREERESTE : boolean;

  begin (* Creereste *)
    creereste := duplicata (expd,expr)
  end; (* Creereste *)

function CREERFACTCOM : boolean;

  begin (* Creerfactcom *)
    creerfactcom := un (expf)
  end; (* Creerfactcom *)


procedure TRTFACT (facteur : refnoeud);

  begin (* Trtfact *)
    expb := base (facteur);
    d := degrequot (expr,expb);
    if d > 0 then
        if  (not expos (fc,expb,d)) then tjrsplace := false
        else
          if (not division (expr,expb,d)) then tjrsplace := false
          else
            if (not prod (expf,expf,fc)) then tjrsplace := false
  end; (* Trtfact *)

begin (* Mee *)
  tjrsplace := true;
   expi := expd;
   if ((not creereste) or (not creerfactcom)) then tjrsplace := false
   else
     begin
       terme := tabnoeud[expi].reffi;
       while ((terme > 0) and (tjrsplace)) do
         begin
           if (tabnoeud [terme].tn <> mu) then trtfact (terme)
           else
             begin
               facteur := tabnoeud [terme].reffi;
               while ((facteur > 0) and tjrsplace) do
                 begin
                   trtfact (facteur);
                   facteur := tabnoeud [facteur].reffr
                 end
             end;
           terme := tabnoeud [terme].reffr
         end
     end;

   if tjrsplace then
     if egalun (expf) then
       begin (* LA MISE EN EVIDENCE A ECHOUE *)
         mee := false; expf := -1
       end
     else
```

```
      if not prod (expf,expf,expr) then
        begin (* PAS ASSEZ DE PLACE POUR MEE *)
          mee := false; expf := 0
        end
      else mee := true
  else
    begin (* PAS ASSEZ DE PLACE POUR MEE *)
      mee := false; expf := 0
    end;
end; (* Mee *)

  begin
  end.

(*----------------------------------------*)
```

```
(*
4.6.3.TEST
      ----*)


(*$S++*)

program TESTMEE;

uses applestuff,
     (*$U #5:GLOBAL.CODE*) global,
     (*$U #5:FILSFRERE.CODE*) sunbrother,
     (*$U #5:LIRE.CODE*) lire,
     (*$U #5:UT.CODE*) utilitaire,
     (*$U #5:MEE.CODE*) miseenevidence,
     (*$U #5:ORD.CODE*) ordonner;

var ch : char;
    ligne : integer;
    veux : boolean;


segment procedure VIDERSTACKS;

  var i : integer;

  begin
    premcar := 1; dercar := nc;
    premnoeud := 1; dernoeud := nn;
    for i := 1 to 82 do chexp[i] := ' ';
    for i := 1 to nc do tabcar[i] := ' '
  end;


segment procedure ECRIRE (exp : integer);

  var i,l,r : integer;

  begin
    writeln('REFERENCE: ',exp);
    r := tabnoeud[exp].refch;
    l := tabnoeud[exp].longch;
    for i:= 1 to l do
      begin
        write (tabcar[r]);
        r := r + 1
      end;
    writeln(' ');
    write ('ORDRINIT: ');
    i := tabnoeud[exp].ordrinit;
    writeln(i)
  end;
```

```
segment procedure PARCOURS (exp : integer);

  var c, crt : integer;
      ch : char;
      fin : boolean;

  begin
    fin := false;
    crt := exp;
    repeat
      page (output);
      writeln ('PERE  : P');
      writeln (' FILS : I');
      writeln (' FRERE: R');
      writeln (' FIN  : F');
      repeat read (ch) until (ch in ['P','I','R','F']);
      page(output);
      gotoxy (0,10);
      case ch of
        'P' : begin
                crt := exp;ecrire (crt); wait
              end;
        'I' : begin
                c := tabnoeud[crt].reffi;
                if (c = 0) then writeln (' PAS DE FILS ')
                else
                  begin
                    if (c < 0) then crt := (-c)
                    else crt := c;
                    ecrire (crt)
                  end;
                wait
              end;
        'R' : begin
                c := tabnoeud[crt].reffr;
                if (c = 0) then writeln (' PAS DE FRERE ')
                else
                  begin
                    if (c < 0) then crt := (-c)
                    else crt := c;
                    ecrire (crt)
                  end;
                wait
              end;
        'F' : fin := true
      end;
    until fin
  end;

segment function SAISIE (var exp :refnoeud) : boolean;

  var s : boolean;

  begin
    repeat
      page (output);
      writeln(' TAPER UNE EXPRESSION ');
      if lirexp (ligne,chexp) then s := true
      else
        begin
          s := false; wait
        end;
```

```
    until s;
    page (output);
    if (not filsfrere (chexp,exp)) then
      begin
        page (output);
        writeln(' PAS ASSEZ DE PLACE POUR FILSFRERE ');
        saisie := false
      end
    else
      begin
        page(output);
        (*
        IF ORDARB (EXP) THEN
          BEGIN *)
            writeln (' ASSEZ DE PLACE POUR FILSFRERE ET ORDARB ');
            saisie := true
          (*END
        ELSE
          BEGIN
            SAISIE := FALSE;
            WRITELN (' PAS ASSEZ DE PLACE POUR ORDARB ')
          END*)
      end;
    wait;page (output)

  end;


segment procedure TESTDEGRE;

  var exp1,exp2 : refnoeud;
      d : integer;

  begin
    if (saisie (exp1) and saisie (exp2)) then
      begin
        page(output);
        d := degrequot (exp1,exp2);
        write('DIVIDENDE :'); ecrire (exp1);
        writeln('DIVISEUR :');ecrire (exp2);
        writeln (' DEGRE : ',d)
      end
  end;


segment procedure TESTEXPOS;

  var puis,exp,n : integer;

  procedure SAISIENBRE (var n : integer);

    begin
      page (output);
      writeln('TAPER UN ENTIER DE 3 CHIFFRES MAXIMUM ');
      read (n)
    end;


  begin
    if saisie (exp) then
      begin
        saisienbre (n);
```

```
        page(output);
        if (not expos (puis,exp,n)) then
          begin
            writeln('PAS ASSEZ DE PLACE POUR EXPOS ');
            writeln ('DERCAR:',dercar,'PREMCAR:',premcar);
            writeln('PREMNOEUD:',premnoeud,'DERNOEUD:',dernoeud)
          end
        else
          begin
            write('BASE:'); ecrire (exp);
            writeln('EXPOSANT:',n);
            write('RESULTAT: ');ecrire (puis);
            wait;
            parcours (puis)
          end
      end
  end;


segment procedure TESTPROD;

  var fact1,fact2,exp : integer;

  begin
    if (saisie (fact1) and saisie(fact2)) then
      begin
        page(output);
        if (not prod (exp,fact1,fact2)) then
          begin
            writeln('PAS ASSEZ DE PLACE POUR PROD ');
            writeln ('DERCAR:',dercar,'PREMCAR:',premcar);
            writeln('PREMNOEUD:',premnoeud,'DERNOEUD:',dernoeud)
          end
        else
          begin
            writeln('FACTEUR1:'); ecrire (fact1);
            writeln('FACTEUR2:'); ecrire (fact2);
            writeln('PRODUIT :'); ecrire (exp);
            wait;
            parcours(exp)
          end
      end
  end;

segment procedure TESTDIVISION;

  var divid,divis,exp,d : integer;

  begin
    if (saisie (divid) and saisie(divis)) then
      begin
        page(output);
        gotoxy (1,10);

        d:= degrequot (divid,divis);
        writeln('DEGREQUOTIENT :',d);
        wait;
        page(output);

        if (not division (divid,divis,d)) then
          begin
            writeln('PAS ASSEZ DE PLACE POUR DVISION ');
```

```
            writeln ('DERCAR:',dercar,'PREMCAR:',premcar);
            writeln('PREMNOEUD:',premnoeud,'DERNOEUD:',dernoeud)
          end
        else
          begin
            writeln(' ');
            writeln('      RESULTAT DIVISION     ');
            writeln('      ******** ********     ');
            writeln(' ');
            writeln('QUOTIENT:'); ecrire (divid);
            wait;
            parcours (divid)
          end
      end
  end;


segment procedure TESTEMEE;

  var expf,exp : integer;

  begin
    if saisie (exp) then
      begin
        page(output);
        if (tabnoeud[exp].tn <> pl) then
          begin
            gotoxy(0,10);
            writeln('      J ATTENDS UNE SOMME ')
          end
        else
          if (not mee (exp,expf)) then

              if expf = 0 then
                begin
                  writeln('PAS ASSEZ DE PLACE POUR MEE ');
                  writeln ('DERCAR:',dercar,'PREMCAR:',premcar);
                  writeln('PREMNOEUD:',premnoeud,'DERNOEUD:',dernoeud)
                end
              else
                if expf = (-1) then writeln('  LA MISE EN EVIDENCE
                                               a echoue ')
                else writeln ('  IL Y A ERREUR SUR EXPF ')

          else
            begin
              page (output);
              gotoxy (13,5);
              writeln('ENONCE:'); ecrire (exp);
              writeln (' ');
              gotoxy (0,15);
              writeln('RESULTAT DE LA MISE EN EVIDENCE :');
              ecrire (expf);
              wait;
              parcours (expf)
            end
    end
  end;
```

```
procedure ETATSTACK;

  begin
    page(output);
    writeln ('   DERCAR: ',dercar,'   PREMCAR:  ',premcar);
    writeln (' ');
    writeln('   PREMNOEUD: ',premnoeud,'   DERNOEUD:  ',dernoeud)
  end;


procedure AFMENU (var ch : char);

  begin
    writeln ('TESTER DEGREQUOT  : 1');
    writeln ('TESTER EXPOS      : 2');
    writeln ('TESTER PROD       : 3');
    writeln ('TESTER DIVISION   : 4');
    writeln ('TESTER MEE        : 5');
    writeln ('FIN               : 6');
    writeln ('VIDER LES STACKS  : 7');
    writeln ('ETATS DES STACKS  : 8');
    repeat read (ch) until (ch in ['1'..'8']);
  end;


begin
  viderstack;
  veux := true;
  ligne := 7;
  page(output);
  repeat
    afmenu (ch);
    case ch of
      '1': testdegre;
      '2': testexpos;
      '3': testprod;
      '4': testdivision;
      '5': testemee;
      '6': veux := false;
      '7': viderstacks;
      '8': etatstack
    end;
    wait;
    page(output)
  until (not veux)
end.
(*-------------------------------------------------------------------*)
```

ANNEXE5 MANUEL UTILISATEUR

1.LISTE DES FICHIERS ".TEXT"

1.GLOBAL.TEXT

Contient les déclarations des variables globales (stacks,...)

2.LIRE

LIRE.TEXT contient le code de la saisie d'une expression à l'écran, et de sa correction syntaxique.
LIRESPEC.TEXT contient les spécifications concrètes du meme unit.
TESTLIRE.TEXT contient le code du programme permettant de tester la saisie d'une expression.

3.FILSFRERE

FILSFRERE.TEXT contient le code la construction de l'arbre fils-frère à partir d'une chaine de caractères.

FFSPEC.TEXT et FFSPEC2.TEXT contiennent les spécifications du meme unit.

TESTFF.TEXT contient le code du programme correspondant.

4.ORDONNER

ORDSPEC.TEXT contient les spécifications du unit d'ordonnancement de l'arbre fils-frère.

ORD.TEXT contient le code de ce unit

5.UTILITAIRES

UTSPEC.TEXT contient les spécifications du unit de comparaison et de construction d'expressions.

UT.TEXT contient le code du meme unit.

TESTUT.TEXT contient le programme de test correspondant aux utilitaires et à l'ordonnancement.

6.MEE

MEESPEC.TEXT contient les spécifications du unit

de mise en évidence et des fonctions DIVISION, DEGREQUOT, EXPOS, PROD.

MEE.TEXT et MEE2.TEXT contiennent le code.

TESTMEE.TEXT contient le programme de test de la mise en évidence et des fonctions ci-dessus.

2.LISTE DES FICHIERS ".CODE"

GLOBAL.CODE

LIRE.CODE

TESTLIRE.CODE

FILSFRERE.CODE

TESTFF.CODE

ORD.CODE

UT.CODE

TESTUT.CODE

MEE.CODE

TESTMEE.CODE

BIBLIOGRAPHIE

Kathleen Jensen, Niklaus Wirth.

PASCAL "USER MANUAL AND REPORT"
Springer-Verlag
New-York Heidelberg Berlin

Apple II

APPLE PASCAL
"OPERATING SYSTEM REFERENCE MANUAL."
Apple Computer Inc.

Jean-Claude SIMON

"L'EDUCATION ET L'INFORMATISATION
DE LA SOCIETE".
Fayard.

Gérard BOSSUET

"L'ORDINATEUR A L'ECOLE".
Puf. L'éducateur.

Edward FEIGENBAUM
Pamela Mc CORDUCK

"LA CINQUIEME GENERATION".
Interéditions.